2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Task-Constrained Motion Planning for Underactuated Robots

Massimo Cefalo, Giuseppe Oriolo

Abstract— **This paper addresses the motion planning problem in the presence of obstacles for underactuated robots that are assigned a geometric task. It is assumed that the robot is subject to kinematic (joint limits, joint velocity bounds) as well as dynamic (torque bounds) constraints. Building on our previous work on task-constrained motion planning, we describe a randomized planner that works directly at the torque level and generates solutions by separating geometric motions from time history. The effectiveness of the proposed approach is shown by planning collision-free swing-up maneuvers for a Pendubot system.**

I. Introduction

Underactuated robots are systems that have more degrees of freedom than actuators. Underactuation arises in many situations: nonprehensile manipulation [1], acrobatic robots [2], legged locomotion [3], surgical robotics [4], free-floating robots [5] and manipulators with passive joints [6]. Other notable examples of underactuated robots are humanoids, underwater vehicles, helicopters, aircrafts and satellites. Underactuation can also emerge in unexpected situations, e.g., in case of actuator failures [7]. Finally, it may be related to the adoption of a specific control strategy without necessarily corresponding to a physical characteristic of the system.

In general, underactuation introduces non-integrable constraints either at the kinematic or dynamic level. For this reason, and because the number of configuration coordinates is larger than the available number of inputs, planning for an underactuated system is a challenging problem, even neglecting the possible presence of obstacles, tasks or other constraints (such as kinematic or dynamic constraints). Indeed, a general planning method for these systems is not available and most of the solutions proposed in the literature address the problem with reference to specific underactuated robots, whose characteristics are exploited. A notable exception are planning approaches based on differential flatness or on dynamic feedback linearizability, which are essentially equivalent properties (see [8] and the references therein for a discussion). These methods are applicable to the whole class of underactuated systems that admit a flat (or linearizing) output, which however does not include all possible robots of interest. For example, the archetypal Acrobot and Pendubot systems do not admit a flat output.

Work on the subject includes [9], that presents a sampling-based motion planner based on the subdivision method for underactuated systems with significant drift term. In [10], a point-to-point motion planning algorithm is proposed for underactuated space robots using high-order polynomials. In [11] an algorithm based on the endogenous configuration space is used to solve the task prioritized motion planning problem. Finally, an approach is proposed in [12] to reduce the complexity of the planning problem for underactuated systems based on an extension of partial feedback linearization control into a task-space framework.

However, the above methods are typically unable to plan motions in the simultaneous presence of workspace obstacles and task constraints; on the other hand, the latter arise in many practical applications. For instance, manipulators used in industrial processes are frequently required to follow specific end-effector paths or trajectories for welding, drawing, cutting or assembling. Another particularly interesting example is the case of manipulation tasks executed by underactuated UAVs (e.g., quadrotor).

This paper considers the motion planning problem in the presence of obstacles for general (i.e., non-flat) underactuated robots that are assigned a certain task. We assume that the robot is subject to kinematic (joint limits, joint velocity bounds) as well as dynamic (torque bounds) constraints. Taking inspiration from our previous work on task-constrained motion planning for fully actuated systems [13], [14], [15], we propose a randomized planner that works directly at the level of torques and generates solutions by separating geometric motions from the time history. The effectiveness of the proposed approach will be shown by planning collision-free swing-up maneuvers for a Pendubot system.

The paper is organized as follows. In Section II we formulate the considered planning problem. Section III discusses the geometric structure of the search space. In Section IV we describe some basic relationships between the underactuated robot dynamics and the task dynamics. The proposed algorithm is described in Section V, and validated in Section VI through planning experiments. Possible future developments are hinted at in the concluding section.

II. Problem Formulation

Consider an underactuated robot, i.e., a mechanical system in which the number of generalized coordinates exceeds that of available control inputs. Its dynamic model can be expressed[1] in the Lagrangian form as

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{pmatrix} \boldsymbol{\tau}_a \\ 0 \end{pmatrix}, \tag{1}$$

The authors are with the Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, Via Ariosto 25, 00185 Rome, Italy. E-mail: {cefalo,oriolo}@diag.uniroma1.it. This work is supported by the EU FP7 ICT-287513 SAPHARI project.

[1] It is always possible to put the model in this form, in which the available generalized forces directly act on a subset of the configuration vector, by a proper definition of generalized coordinates and/or an input transformation.

where $\boldsymbol{q} \in \mathcal{C}$ is the configuration n-vector, $\boldsymbol{B}(\boldsymbol{q})$ is the $n \times n$ inertia matrix, $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the n-vector of Coriolis, centrifugal and potential forces, and $\boldsymbol{\tau}_a$ are the $p < n$ generalized forces, simply referred to as *torques* in the following.

We assume that the robot is subject to:

- joint range limits $\boldsymbol{q}_m \leq \boldsymbol{q} \leq \boldsymbol{q}_M$;
- maximum bounds $|\dot{\boldsymbol{q}}| \leq \dot{\boldsymbol{q}}_M$ on generalized velocities;
- maximum bounds $|\boldsymbol{\tau}_a| \leq \boldsymbol{\tau}_{aM}$ on torques.

Any trajectory in $\mathcal{C}$ that can be followed by the robot (in spite of its underactuation) and at the same time satisfies all the above constraints will be called *feasible*.

The robot moves in a 3D workspace containing obstacles. To keep the notation light, we assume that the obstacles are fixed, but the proposed planner works unchanged in the case of obstacles that move along known trajectories, as in [15]. Denote by $\mathcal{R}(\boldsymbol{q})$ and $\mathcal{O}$, respectively, the workspace volume occupied by the robot at $\boldsymbol{q}$ and by the obstacles.

The robot must perform a task described by an m-dimensional vector $\boldsymbol{y}$, with an associated task space $\mathcal{Y}$. The task coordinates are related to the configuration coordinates via a kinematic map $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{q})$, whose differential version is

$$\dot{\boldsymbol{y}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}},$$

with $\boldsymbol{J} = \partial \boldsymbol{f} / \partial \boldsymbol{q}$ the $m \times n$ task Jacobian. For obvious reasons, which will be formally discussed in the following, we will assume that $m \leq p$; i.e., the dimension of the task is not larger than the number of control inputs.

Suppose that a desired *path* $\boldsymbol{y}_d(s) \in \mathcal{Y}$ is assigned for the task coordinates, with $s \in [s_i, s_f]$ a path parameter. The considered problem consists in planning a *trajectory* in $\mathcal{C}$ that is feasible, collision-free, and realizes the desired task path. We shall obtain such a trajectory as the composition of two parts, i.e., a path $\boldsymbol{q}(s) \in \mathcal{C}$, $s \in [s_i, s_f]$, and a continuous time history $s(t) : [0, T] \mapsto [s_i, s_f]$, such that :

1. $\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{q}(s(t))) = \boldsymbol{y}_d(s(t))$, $\forall t \in [0, T]$
   % task is always on the assigned path
2. $s(0) = s_i$ and $s(T) = s_f$
   % the whole task path is realized
3. $\boldsymbol{q}(0) = \boldsymbol{q}_i$, $\dot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}(T) = \boldsymbol{0}$
   % assigned initial configuration is matched
   % initial and final velocities are zero
4. $\boldsymbol{q}_m \leq \boldsymbol{q}(t) \leq \boldsymbol{q}_M$, $|\dot{\boldsymbol{q}}(t)| \leq \dot{\boldsymbol{q}}_M$ and $|\boldsymbol{\tau}_a(t)| \leq \boldsymbol{\tau}_{aM}$, $\forall t \in [0, T]$
   % joint limits, velocity and torque bounds are satisfied
5. $\mathcal{R}(\boldsymbol{q}(s(t))) \cap \mathcal{O} = \emptyset$, $\forall t \in [0, T]$
   % collisions with obstacles are avoided

While the initial configuration $\boldsymbol{q}_i$ is given, the final configuration $\boldsymbol{q}(s_f) = \boldsymbol{q}(T)$ and the duration T of the plan will be generated by the planner. If $\boldsymbol{q}(T)$ is an unforced equilibrium, setting $\dot{\boldsymbol{q}}(0)$ and $\dot{\boldsymbol{q}}(T)$ to zero produces a rest-to-rest motion.

Note that the time history $s(t)$ is not required to be monotonic: i.e., s may increase or decrease over time, corresponding respectively to a *forward* or a *backward* motion along the task path. The planner may exploit such possibility to achieve certain tasks in spite of its limited dynamic capabilities (underactuation, velocity and torque bounds).

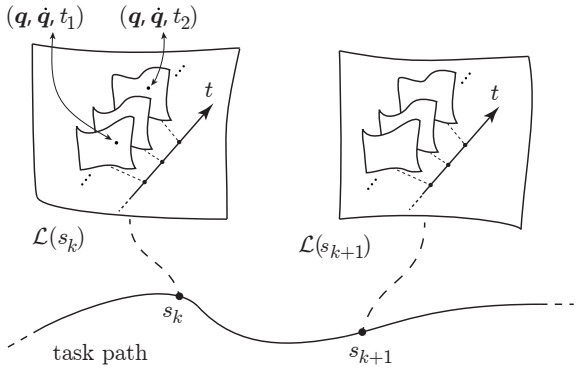

Fig. 1. $\mathcal{S}_{\text{task}}$ is a foliation: each leaf $\mathcal{L}(s)$ is the set of points $(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \in \mathcal{S}$ such that $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are consistent with the task path constraint for a certain value of s, while t may assume any value.

III. Search Space

Define the state space of the robot as $\mathcal{X} = \mathcal{C} \times T_{\boldsymbol{q}}\mathcal{C}$, where $T_{\boldsymbol{q}}\mathcal{C}$ denotes the tangent space of $\mathcal{C}$ at $\boldsymbol{q}$. In itself, $\mathcal{X}$ does not adequately reflect the limitations of the considered system. In fact, due to the underactuation, the joint limits, and the velocity bounds, a state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ may or not be *reachable* by the robot; and even when it is reachable, it may be reachable at a certain time instant and not at another due to the existence of torque bounds. To account for this, we first augment states with time, and define

- the *state-time space* as $\mathcal{S} = \mathcal{X} \times [0, \infty)$;
- the *occupied state-time space* as $\mathcal{S}_{\text{occ}} = \{(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \in \mathcal{S} : \mathcal{R}(\boldsymbol{q}(t)) \cap \mathcal{O} \neq \emptyset\}$;
- the *free state-time space* as $\mathcal{S}_{\text{free}} = \mathcal{S} \setminus \mathcal{S}_{\text{occ}}$.

Then, we call $\bar{\mathcal{S}}_{\text{free}}$ (*reachable free state-time space*) the subset of $\mathcal{S}_{\text{free}}$ that is actually reachable. Trajectories in $\bar{\mathcal{S}}_{\text{free}}$ are by definition feasible and collision-free.

The existence of a task path constraint further reduces the region of $\mathcal{S}$ where to look for a solution. Define the *task-constrained state-time space* as the set of points of $\mathcal{S}$ where the state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is consistent with the assigned task path:

$$\mathcal{S}_{\text{task}} = \{(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \in \mathcal{S} : \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{y}_d(s), \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{y}'_d(s)\dot{s}, \text{ for some } s \in [s_i, s_f], \dot{s} \in (-\infty, \infty)\},$$

having used the notation $()' = d()/ds$. Note that t is immaterial in $\mathcal{S}_{\text{task}}$: for any point in $\mathcal{S}_{\text{task}}$, there exist an infinity of other points with the same state and different time instant $t \in [0, \infty)$. From a geometric viewpoint, $\mathcal{S}_{\text{task}}$ is a manifold with boundary which foliates:

$$\mathcal{S}_{\text{task}} = \cup_{s \in [s_i, s_f]} \mathcal{L}(s)$$

with each *leaf* $\mathcal{L}(s)$ associated to a value of $s \in [s_i, s_f]$:

$$\mathcal{L}(s) = \{(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \in \mathcal{S} : \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{y}_d(s), \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{y}'_d(s)\dot{s} \text{ for some } \dot{s} \in (-\infty, \infty)\}.$$

Figure 1 illustrates the structure of $\mathcal{S}_{\text{task}}$.

The existence of a solution to the considered planning problem depends on the interplay between the assigned task and the various constraints acting on the robot, and in particular on the connectedness of the search space $\bar{\mathcal{S}}_{\text{free}} \cap \mathcal{S}_{\text{task}}$.

IV. GEOMETRIC TASK DYNAMICS

The objective of this section is to derive some basic relationships that will be used in the motion generation part of the proposed planner. In particular, we wish to characterize the geometric task accelerations that can be produced as a result of the available torque input $\boldsymbol{\tau}_a$.

First, in view of the separation between geometric motion and time history, generalized velocities and accelerations along a certain solution may be expressed respectively as

$$\dot{\boldsymbol{q}}(t) = \dot{s}(t)\boldsymbol{q}'(s) \tag{2}$$

and

$$\ddot{\boldsymbol{q}}(t) = \ddot{s}(t)\boldsymbol{q}'(s) + \dot{s}^2(t)\boldsymbol{q}''(s), \tag{3}$$

where $\boldsymbol{q}' = d\boldsymbol{q}/ds$ and $\boldsymbol{q}'' = d\boldsymbol{q}'/ds$ are respectively the geometric generalized velocities and accelerations.

Substituting (2) and (3) in (1), we can write the dynamic equations of the robot in a form that separates the geometric motion from the time history:

$$\boldsymbol{B}(\boldsymbol{q})(\ddot{s}(t)\boldsymbol{q}'(s) + \dot{s}^2(t)\boldsymbol{q}''(s)) + \boldsymbol{n}(\boldsymbol{q}, \dot{s}(t)\boldsymbol{q}'(s)) = \begin{pmatrix}\boldsymbol{\tau}_a \\ 0\end{pmatrix},$$

compactly rewritten as

$$\widetilde{\boldsymbol{B}}(\boldsymbol{q}, \dot{s})\boldsymbol{q}'' + \boldsymbol{\varphi}(\boldsymbol{q}, \boldsymbol{q}', \dot{s}, \ddot{s}) = \begin{pmatrix}\boldsymbol{\tau}_a \\ 0\end{pmatrix}, \tag{4}$$

where

$$\widetilde{\boldsymbol{B}}(\boldsymbol{q}, \dot{s}) = \dot{s}^2\boldsymbol{B}(\boldsymbol{q})$$

and

$$\boldsymbol{\varphi}(\boldsymbol{q}, \boldsymbol{q}', \dot{s}, \ddot{s}) = \ddot{s}\boldsymbol{B}(\boldsymbol{q})\boldsymbol{q}' + \boldsymbol{n}(\boldsymbol{q}, \dot{s}(t)\boldsymbol{q}'(s)).$$

If $\dot{s} \neq 0$ [2], matrix $\widetilde{\boldsymbol{B}}$ is invertible and from (4) we can obtain:

$$\boldsymbol{q}'' = \widetilde{\boldsymbol{B}}^{-1}(\boldsymbol{q}, \dot{s}) \left(\begin{pmatrix}\boldsymbol{\tau}_a \\ 0\end{pmatrix} - \boldsymbol{\varphi}(\boldsymbol{q}, \boldsymbol{q}', \dot{s}, \ddot{s}) \right). \tag{5}$$

This formula encodes the robot underactuation, i.e., it represents all geometric generalized accelerations that can be produced at a given state by virtue of the available input $\boldsymbol{\tau}_a$.

Consider now the second-order kinematic map that expresses the geometric acceleration of the task variables:

$$\boldsymbol{y}'' = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{q}'' + \boldsymbol{J}'(\boldsymbol{q}, \boldsymbol{q}')\boldsymbol{q}'. \tag{6}$$

Substituting (5) in (6) we obtain

$$\boldsymbol{y}'' = \boldsymbol{A}(\boldsymbol{q}, \dot{s}) \left(\begin{pmatrix}\boldsymbol{\tau}_a \\ 0\end{pmatrix} - \boldsymbol{\varphi}(\boldsymbol{q}, \boldsymbol{q}', \dot{s}, \ddot{s}) \right) + \boldsymbol{J}'(\boldsymbol{q}, \boldsymbol{q}')\boldsymbol{q}', \tag{7}$$

where

$$\boldsymbol{A}(\boldsymbol{q}, \dot{s}) = \boldsymbol{J}(\boldsymbol{q})\widetilde{\boldsymbol{B}}^{-1}(\boldsymbol{q}, \dot{s}).$$

Since the robot is underactuated it is $p < n$, and we can partition the $m \times n$ matrix $\boldsymbol{A}$ as $(\boldsymbol{A}_a \quad \boldsymbol{A}_u)$, where $\boldsymbol{A}_a$ is an $m \times p$ submatrix. Therefore we can write

$$\boldsymbol{y}'' = \boldsymbol{A}_a(\boldsymbol{q}, \dot{s})\boldsymbol{\tau}_a - \boldsymbol{A}(\boldsymbol{q}, \dot{s})\,\boldsymbol{\varphi}(\boldsymbol{q}, \boldsymbol{q}', \dot{s}, \ddot{s}) + \boldsymbol{J}'(\boldsymbol{q}, \boldsymbol{q}')\boldsymbol{q}'. \tag{8}$$

Equation (8) is the task-level counterpart of (5), in that it represents all geometric task accelerations that can be produced at a given state by virtue of $\boldsymbol{\tau}_a$.

[2] This condition is not necessary for the planner; see footnote 4.

V. PROPOSED PLANNER

The proposed planner expands a tree in $\bar{\mathcal{S}}_{\text{free}} \cap \mathcal{S}_{\text{task}}$. A predefined sequence $\{s_1 = s_i, \ldots, s_k, \ldots, s_N = s_f\}$ of values of s is used to extract N samples of the assigned task path, denoted by $\boldsymbol{y}_k = \boldsymbol{y}_d(s_k)$, $k = 1, \ldots, N$. These samples will be used to bias the search. Let $\mathcal{L}_k = \mathcal{L}(s_k)$ be the leaf associated to $\boldsymbol{y}_k$ (see Figure 1).

Each vertex of the tree is a triplet $(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)$ representing a robot state and the time at which it was attained. An edge is a feasible subtrajectory joining two vertices that lie on adjacent leaves, and is produced by a specialized motion generation scheme. The tree is rooted at $(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i, 0)$, i.e., at the initial state and time. At any iteration of the planner, the root will be the only vertex on the first leaf $\mathcal{L}_1$, whereas any other leaf $\mathcal{L}_k$, $k > 1$, may contain several vertexes.

A. Motion Generation

At the heart of our proposed planner is a motion generation scheme that can be invoked from a generic vertex of the tree located on a certain leaf, producing a subtrajectory that is contained in $\bar{\mathcal{S}}_{\text{free}} \cap \mathcal{S}_{\text{task}}$ and lands on either the next or the previous leaf. A new vertex is generated at the landing state and time, provided that the subtrajectory is collision-free. Due to the presence of underactuation and torque constraints, the motion generation scheme must operate at the torque level. To this end, we shall use the basic relationships established in the previous section.

Consider a generic vertex $V = (\boldsymbol{q}_V, \dot{\boldsymbol{q}}_V, t_V)$ on the leaf $\mathcal{L}_k$. All vertexes on $\mathcal{L}_k$ share the same value of $s = s_k$, whereas the value of $\dot{s} = \dot{s}_V$ is different for each V, as a byproduct of the subtrajectory that generated that vertex. Equation (8) indicates that, once the value of $\ddot{s}$ has been chosen, the value of the torque τ_a can be derived from the desired geometric task acceleration.

In particular, we choose $\ddot{s}$ at V as

$$\ddot{s} = \ddot{s}_V, \tag{9}$$

with $\ddot{s}_V$ a constant value chosen within a predefined range $[-c_{\max}, c_{\max}]$. This means that, from t_V on, the dependence of s on t will be quadratic. A simple reasoning shows that, depending on the value of $\dot{s}_V$ and of the chosen $\ddot{s}_V$, one can obtain four kinds of motions of s over t, and correspondingly of $\boldsymbol{y}(s)$ over $\boldsymbol{y}_d(s)$: (1) a monotonic forward motion from s_k to s_{k+1} (2) a motion which moves initially backward from s_k but then reverses its direction before s_{k-1} and proceeds forward to reach s_{k+1} (3) a monotonic backward motion from s_k to s_{k-1} (4) a motion which moves initially forward from s_k but then reverses its direction before s_{k+1} and proceeds backwards to reach s_{k-1}.

In any case, once $\ddot{s}$ has been chosen, we can compute the torque vector $\boldsymbol{\tau}_a$ for realizing a certain $\boldsymbol{y}_d''$ by inverting (8):

$$\boldsymbol{\tau}_a = \boldsymbol{A}_a^{\dagger}(\boldsymbol{y}_d'' + \boldsymbol{K}_p\boldsymbol{e}_y + \boldsymbol{K}_d\boldsymbol{e}_y' - \boldsymbol{J}'\boldsymbol{q}' + \boldsymbol{A}\boldsymbol{\varphi}) + (\boldsymbol{I} - \boldsymbol{A}_a^{\dagger}\boldsymbol{A}_a)\boldsymbol{\tau}_V \tag{10}$$

where $\boldsymbol{A}_a^{\dagger}$ is the pseudoinverse of $\boldsymbol{A}_a$, $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are positive definite matrices, $\boldsymbol{e}_y = \boldsymbol{y}_d - \boldsymbol{y}$ is the task error, $\boldsymbol{e}_y'$ its geometric derivative, and $\boldsymbol{\tau}_V$ is an p-dimensional *residual*

torque vector which can be arbitrarily specified without affecting the task (in fact, $\boldsymbol{I} - \boldsymbol{A}_a^\dagger \boldsymbol{A}_a$ is the orthogonal projection matrix in the null space of $\boldsymbol{A}_a$). In the above formula, all dependencies (including that of $\boldsymbol{\varphi}$ on $\ddot{s}$) have been dropped for compactness. One may easily verify that the above choice of $\boldsymbol{\tau}_a$ yields exponentially stable tracking of the desired task path.

By assumption[3] $m \leq p$, thus it is $\boldsymbol{A}_a^\dagger = \boldsymbol{A}_a^T (\boldsymbol{A}_a \boldsymbol{A}_a^T)^{-1}$ provided that $\boldsymbol{A}_a$ is full row rank. A necessary condition for the latter condition to hold is that $\boldsymbol{J}$ is nonsingular[4]. Note that the nonsingularity of $\boldsymbol{J}$ only guarantees that $\boldsymbol{A}$ is full row rank, while its submatrix $\boldsymbol{A}_a$ may not be such. When this happens, the desired task geometric acceleration is not realizable at the current state due to underactuation. Note also that in the limit case $m = p$, we obtain $\boldsymbol{A}_a^\dagger = \boldsymbol{A}_a^{-1}$ and $\boldsymbol{I} - \boldsymbol{A}_a^\dagger \boldsymbol{A}_a = \boldsymbol{0}$ (no actual redundancy).

Motion is then generated by integrating eqs. (1) and (10) from vertex V. In doing so, velocity and torque bounds are continuously verified, together with absence of collisions. If either of these is violated, or if $\boldsymbol{A}_a$ loses rank, motion generation is prematurely terminated. Otherwise, integration stops when the subtrajectory lands on an adjacent leaf to $\mathcal{L}_k$, be it $\mathcal{L}_{k+1}$ or $\mathcal{L}_{k-1}$.

It should be noted that the choice of $\ddot{s}$ in the last integration interval $[s_{N-1}, s_N]$ is not arbitrary: in fact, to guarantee that the robot velocity is zero at the final point of the task, we must impose that $\dot{s}(t_N) = 0$. An easy computation shows that this is obtained by letting

$$\ddot{s}_N = -\frac{\dot{s}_{N-1}^2}{2(s_N - s_{N-1})}. \tag{11}$$

As an alternative to choosing a $\ddot{s}$ which is constant throughout the integration interval as in (9), and then having to check the torque bounds, one may use a piecewise-constant $\ddot{s}$, with the value of each piece chosen in such a way that the torque bound is guaranteed to be satisfied.

In particular, let $\boldsymbol{w} = \boldsymbol{y}_d'' + \boldsymbol{K}_p \boldsymbol{e}_y + \boldsymbol{K}_d \boldsymbol{e}_y' - \boldsymbol{J}' \boldsymbol{q}'$ and $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{A}_a^\dagger \boldsymbol{A}_a$ for compactness. If

$$|\ddot{s}| \leq \min_{i \in [1,\ldots,p]} \frac{\tau_{aM,i} - |\boldsymbol{A}_a^\dagger \boldsymbol{w} + \boldsymbol{A}_a^\dagger \boldsymbol{A} \boldsymbol{n} + \boldsymbol{P} \boldsymbol{\tau}_V|_i}{|\boldsymbol{A}_a^\dagger \boldsymbol{A} \boldsymbol{B} \boldsymbol{q}'|_i} \tag{12}$$

where $|\cdot|_i$ indicates the modulus of the i-th component of a vector, it can be easily verified from (10) that $|\boldsymbol{\tau}_a| \leq \boldsymbol{\tau}_{a,M}$.

B. Tree Expansion

The planning tree is expanded using an RRT-like mechanism. At each iteration, a random task sample $\boldsymbol{y}_{\mathrm{rand}} = \boldsymbol{y}_k$, with $k \in \{1, \ldots, N\}$, is extracted from the predefined sequence, and an inverse solution $\boldsymbol{q}_{\mathrm{rand}} = \boldsymbol{f}^{-1}(\boldsymbol{y}_{\mathrm{rand}})$ is computed. A random task-consistent generalized velocity $\dot{\boldsymbol{q}}_{\mathrm{rand}} \in [-\dot{\boldsymbol{q}}_M, \dot{\boldsymbol{q}}_M]$ is chosen and attached to $\boldsymbol{q}_{\mathrm{rand}}$. Finally, a time instant t_{rand} is sampled from $[0, t_{\max}]$, with $t_{\max}$ the largest time instant associated to a vertex in the current tree. By construction, $(\boldsymbol{q}_{\mathrm{rand}}, \dot{\boldsymbol{q}}_{\mathrm{rand}}, t_{\mathrm{rand}})$ is a sample of $\mathcal{S}_{\mathrm{task}}$.

At this point, the tree is searched for the closest vertex to $(\boldsymbol{q}_{\mathrm{rand}}, \dot{\boldsymbol{q}}_{\mathrm{rand}}, t_{\mathrm{rand}})$, according to a suitably defined metric[5]. Denote this vertex by $(\boldsymbol{q}_{\mathrm{near}}, \dot{\boldsymbol{q}}_{\mathrm{near}}, t_{\mathrm{near}})$, and say it is located on a generic leaf $\mathcal{L}_k$.

Then, the tree is expanded from $V = (\boldsymbol{q}_{\mathrm{near}}, \dot{\boldsymbol{q}}_{\mathrm{near}}, t_{\mathrm{near}})$ using the previous motion generation scheme, in which the residual torque vector $\boldsymbol{\tau}_V$ is chosen randomly. As explained before, as soon as one of the two adjacent leaves $\mathcal{L}_{k+1}$ or $\mathcal{L}_{k-1}$ is reached by a feasible, collision-free subtrajectory, a new vertex is placed at the landing point. As a byproduct of the integration procedure, we obtain the time instant associated to the new vertex. Whenever a subtrajectory is discarded due to constraint violation, a new tree expansion takes place.

An alternative to a pure random choice of $\boldsymbol{\tau}_V$ is to evaluate the effect of a finite number of candidate choices (e.g., a predefined set of primitives, or of randomly chosen values) and to select the one that produces the final vertex closest to $(\boldsymbol{q}_{\mathrm{rand}}, \dot{\boldsymbol{q}}_{\mathrm{rand}}, t_{\mathrm{rand}})$.

A final remark is in order concerning the discrete sequence $\{s_1, \ldots, s_N\}$ used by the algorithm. While its introduction simplifies the description, it is by no means necessary. Indeed, one may extract random values of s from the whole continuous interval $[s_i, s_f]$ to generate the task sample $\boldsymbol{y}_{\mathrm{rand}}$. The length of the task path portion covered along the subtrajectory generated from $\boldsymbol{q}_{\mathrm{near}}$ can be then decided using some other criteria (e.g., fixed, or randomly generated itself).

VI. Planning Experiments

In this section, we report results of planning experiments for the Pendubot system, a planar manipulator with two rotational joints that moves in the vertical plane and is equipped with a single motor at the first joint. In particular, we have used the dynamic parameters of the prototype available in our lab and shown in Fig. 2: the first link has length 0.148 m and mass 0.19 Kg, while the second link has length 0.181 m and mass 0.07 Kg. Thanks to the mechanical design, the joints can rotate indefinitely; also, the velocity bounds (related for the first link to the maximum rotational speed of the motor and for the second link to the resolution of the encoder) are very high (around 250 rad/s for both) and in practice irrelevant for the planner with respect to the torque limits.

Although the Pendubot is a relatively simple underactuated system, it is known to be particularly challenging from the viewpoint of planning because it is not linearizable via feedback (equivalently, it admits no flat outputs). As a consequence, motions for this robot are typically generated as byproduct of feedback controllers, that however disregard the presence of obstacles, velocity and torque bounds.

We implemented the proposed planner for the Pendubot as a C++ library for Kite, a software development kit

[3] In the case $m > p$, matrix $\boldsymbol{A}_a$ would be a 'tall' matrix, and therefore equation (8) would admit no solution in general.

[4] Using eq. (7), it may be easily shown that $\boldsymbol{A}_a^\dagger = \dot{s}^2 \tilde{\boldsymbol{A}}_a^\dagger$, where $\tilde{\boldsymbol{A}}_a^\dagger$ does not depend on $\dot{s}$; therefore, condition $\dot{s} \neq 0$ is *not* necessary for $\boldsymbol{A}_a$ to be full row rank.

[5] In particular, a weighted sum is used to characterize distances in the state-time space $\mathcal{S}$.

Fig. 2. The Pendubot available in our laboratory and used in our planning experiments.

for motion planning currently marketed by Siemens. The hardware platform was a 64-bit Intel Core i5-2320 CPU running at 3 GHz.

In both planning experiments, we address the classical *swing-up* problem, which requires to bring the robot from the stable down-down equilibrium to the unstable up-up equilibrium, starting and ending at zero velocity. Interestingly, such a state-to-state planning problem admits a description in the form of a lower-dimensional task. In particular, denoting by y the vertical coordinate of the robot tip, and placing the origin $y = 0$ at the height of the axis of rotation of the first joint, the desired task path can be specified by a function of s that goes from $-\ell$ to ℓ, where ℓ is the sum of the lengths of the two links. In particular, we have used $y(s) = \ell(2s^3 - 1)$, for $s \in [0, 1]$, and extracted a sequence of $N = 11$ equispaced samples from this path (including the endpoints). We have set $K_p = K_d = 10$ in the motion generation scheme, and performed numerical integration using Euler method with step size 0.002 s.

Note that in this planning problem we have $p = m = 1$, i.e., a single actuator and a one-dimensional task. Therefore, there is no redundancy in the torque generation using (10).

In the first planning experiment (Fig. 3) we have considered the actual torque limit of our prototype, i.e. $\tau_M = 1.58$ Nm. An obstacle has been placed very close to the up-up configuration, so that the robot can only execute the swing-up maneuver in the clockwise direction, and in addition the motion of the first link must be carefully planned so as to avoid the obstacle. For this planning scenario we have adopted the generation of $\ddot{s}$ based on eq. (9)), checking then the admissibility of the obtained torque. The planner was able to compute the solution shown in Fig. 3 in about 15 seconds (average over 10 repeated experiments). The swing-up is achieved in 0.62 seconds. Figure 4 confirms that the torque required to perform the planned motion always complies with the actuator bound, whereas Fig. 5 shows the time history generated by the planner as a consequence of the chosen $\ddot{s}$. Note how the final value of $\dot{s}$ is zero, confirming that the

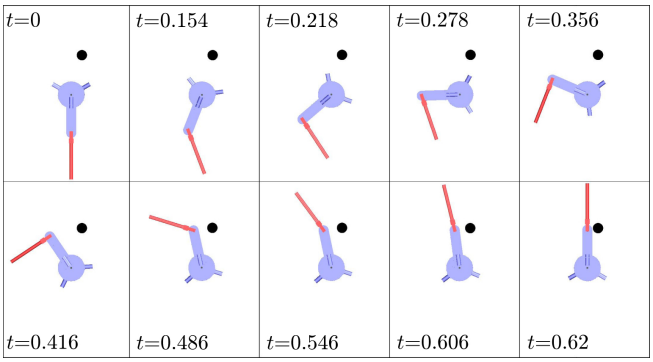

Fig. 3. Planning experiment 1: some snapshots from a solution.

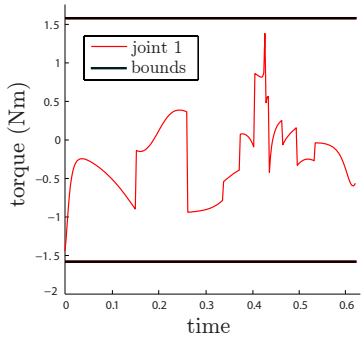

Fig. 4. Planning experiment 1: Required torque along the solution.

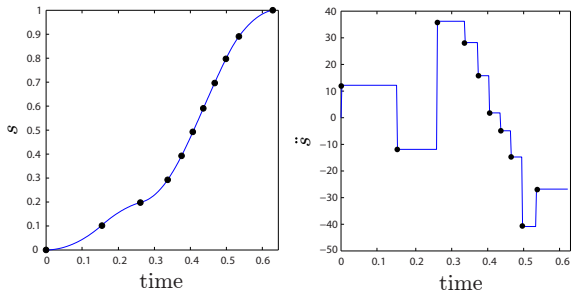

Fig. 5. Planning experiment 1: Time history $s(t)$ and $\ddot{s}(t)$. Black dots are in correspondence of the time instants at which the robot configuration reaches a leaf associated to one of the N samples of the assigned task.

robot will stop at the up-up equilibrium.

In the second planning experiment we have considered a more powerful joint actuator whose torque bound is at $\tau_M = 7.9$ Nm. Also, the obstacle has been moved to the right to give the planner more freedom in generating movements. Finally, we have applied the alternative strategy (12) for choosing $\ddot{s}$ in such a way that the torque bound is certainly satisfied. The average time needed to compute a solution for this scenario is roughly the same of the previous scenario. One such solution is shown in Fig. 6, and achieves swing-up in 0.44 seconds — a shorter time thanks to the increased actuator capability. Note also how the first link can now safely perform a wider swing than in Fig. 6. As before, Figure 7 confirms that the required torque is always within the new bound, whereas Fig. 5 shows the time history

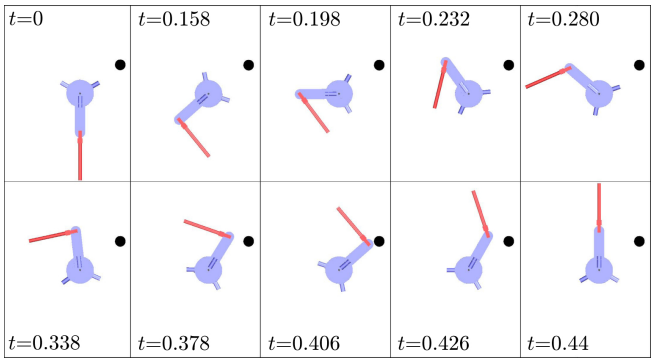

Fig. 6. Planning experiment 2: some snapshots from a solution.

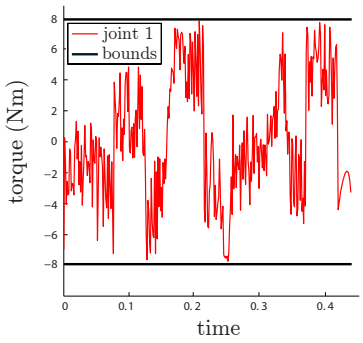

Fig. 7. Planning experiment 2: Required torque along the solution.

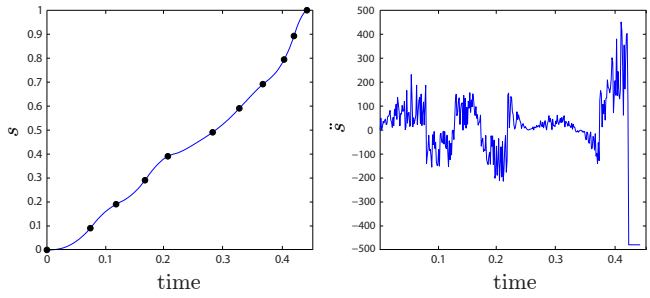

Fig. 8. Planning experiment 2: Time history $s(t)$ and $\ddot{s}(t)$. Black dots are in correspondence of the time instants at which the robot configuration reaches a leaf associated to one of the N samples of the assigned task.

generated by the planner as a consequence of the chosen $\ddot{s}$. Again, the value of $\dot{s}$ at the end of the motion is zero, so that the robot stops at the up-up equilibrium.

The video attachment to the paper contains clips of the generated swing-up motions. Performance details are collected in Table I.

VII. Conclusions

We have presented a randomized motion planning algorithm for underactuated robots in the presence of task path constraints, kinematic constraints and torque bounds. To account for the underactuation and the torque bounds, the proposed planner must necessarily work at the level of torques. The core of the planner is a motion generation scheme inspired by our previous works on task-constrained motion planning. The effectiveness of the proposed planner has been shown by planning swing-up motions for the Pendubot system.

TABLE I

exp	exec time	vertexes	coll checks	duration
1	15.8 s	428	36816	0.62 s
2	10.2 s	268	14102	0.44 s

The Pendubot case study suggests that the proposed technique can be used to transfer underactuated robots between equilibrium points by an appropriate definition of the task variables. We will analyze this issue in more detail to identify other relevant examples. Moreover, we will address the problem of modifying the planner so as to allowing transfers between forced equilibria.

Finally, we intend to apply the proposed planner to higher-dimensional underactuated systems. For example, we will consider manipulation tasks executed by an underactuated UAV (e.g., a quadrotor) equipped with an effector arm.

References

[1] K. Lynch and M. Mason, "Dynamic nonprehensile manipulation: Controllability, planning, and experiments," *Int. J. of Robotics Research*, vol. 18, pp. 64–92, 1999.

[2] J. Nakanishi, T. Fukuda, and D. Koditschek, "A brachiating robot controller," *IEEE Trans. on Robotics and Automation*, vol. 16, pp. 109–123, 2000.

[3] M. Spong, "Bipedal locomotion, robot gymnastics, and and robot air hockey: A rapprochement," in *In: TiTech COE/Super MechanoSystems Workshop 99*, 1999, pp. 34–41.

[4] J. Funda, R. Taylor, B. Eldridge, S. Gomory, and K. Gruben, "Constrained cartesian motion control for teleoperated surgical robots," *IEEE Trans. on Robotics and Automation*, vol. 12, pp. 453–465, Jun 1996.

[5] N. Faiz and S. Agrawal, "Optimal planning of an under-actuated planar body using higher-order method," in *1998 IEEE Int. Conf. on Robotics and Automation*, vol. 1, May 1998, pp. 736–741.

[6] G. Oriolo and Y. Nakamura, "Control of mechanical systems with second-order nonholonomic constraints: underactuated manipulators," in *Decision and Control, 1991, Proceedings of the 30th IEEE Conference on*, Dec 1991, pp. 2398–2403.

[7] H. Arai and S. Tachi, "Position control of manipulator with passive joints using dynamic coupling," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 4, pp. 528–534, Aug 1991.

[8] A. De Luca and G. Oriolo, "Trajectory planning and control for planar robots with passive last joint," *Int. J. of Robotics Research*, vol. 21, pp. 575–590, 2002.

[9] A. Ladd and L. Kavraki, "Motion planning in the presence of drift, underactuation and discrete system changes," in *2005 Robotics: Science and Systems*, Cambridge, USA, 2005.

[10] I. Tortopidis and E. Papadopoulos, "On point-to-point motion planning for underactuated space manipulator systems," *Robotics and Autonomous Systems*, vol. 55, pp. 122–131, 2007.

[11] A. Ratajczak, J. Karpinska, and K. Tchon, "Task-priority motion planning of underactuated systems: an endogenous configuration space approach," *Robotica*, vol. 28, pp. 885–892, 2010.

[12] A. Shkolnik and R. Tedrake, "High-dimensional underactuated motion planning via task space control," in *2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Nice, France, 2008, pp. 3762–3768.

[13] 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*, St. Louis, MO, 2009, pp. 297–302.

[14] M. Cefalo, G. Oriolo, and M. Vendittelli, "Task-constrained motion planning with moving obstacles," in *2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Tokyo, Japan, 2013, pp. 5758–5763.

[15] M. Cefalo and G. Oriolo, "Dynamically feasible task-constrained motion planning with moving obstacles," in *2014 IEEE Int. Conf. on Robotics and Automation*, Hong Kong, China, 2014, pp. 2045–2050.