IEEE Robotics and Automation Letters (RAL) paper presented at the
2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

An Opportunistic Strategy for Motion Planning in the Presence of Soft Task Constraints

Massimo Cefalo, Paolo Ferrari, Giuseppe Oriolo

***Abstract*— Consider the problem of planning collision-free motions for a robot that is assigned a *soft* task constraint, i.e., a desired path in task space with an associated error tolerance. To this end, we propose an opportunistic planning strategy in which two subplanners take turns in generating motions. The *hard planner* guarantees exact realization of the desired task path until an obstruction is detected in configuration space; at this point, it invokes the *soft planner*, which is in charge of exploiting the available task tolerance to bypass the obstruction and returning control to the hard planner as soon as possible. As a result, the robot will perform the desired task for as long as possible, and deviate from it only when strictly needed to avoid a collision. We present several planning experiments performed in V-REP for the PR2 mobile manipulator in order to show the effectiveness of the proposed planner.**

I. Introduction

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

Tasks are generally expressed in terms of a certain set of coordinates, called task coordinates. These may describe quantities related to manipulation (end-effector position and/or orientation), navigation (position of a representative point of the robot, e.g., the center of mass), or perception (placement of sensors in the workspace or directly of features in sensing space, as in visual servoing). Often, the task is assigned as a desired path or trajectory for the task coordinates, resulting in an equality (*hard*) constraint. However, in industrial and service applications there are many situations where the task is better expressed by an inequality (*soft*) constraint; for example, this constraint may represent the fact that the desired path assigned to the task coordinates comes with a certain error tolerance. Figure 1 shows an example of such scenario.

The problem of generating collision-free robot motions in the presence of hard task constraints is known as Task-Constrained Motion Planning (TCMP). In the literature, there exist two main classes of methods for solving the TCMP problem, i.e., optimization- and sampling-based methods.

Optimization-based methods (see [1] for a general review) cast the TCMP problem in the framework of kinematic control, also called redundancy resolution, with the possible inclusion of specific equality or inequality constraints related to the assigned task [2]. The discrete optimization technique presented in [3] is able to compute very accurate tracking; however, avoidance of workspace obstacles is not considered. In any case, it should be kept in mind that, independently of the specific version, kinematic control is a greedy strategy whose optimization capabilities are inherently local; as a consequence, it can work occasionally but never guarantee completeness (finding a solution whenever one exists).

The authors are with the Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, Via Ariosto 25, 00185 Rome, Italy. E-mail: *lastname*@diag.uniroma1.it.

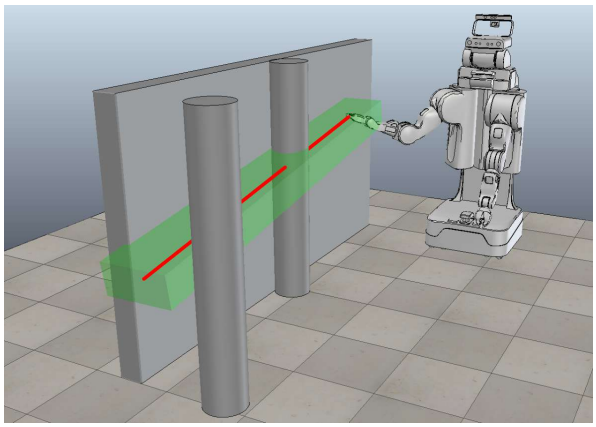

Fig. 1. In the problem of interest, the robot is assigned a soft task specified by a desired path (red line) and a tolerance (green volume).

Sampling-based methods for solving the TCMP problem typically use a mechanism for projecting configuration space samples on the submanifold where the task constraint is satisfied; see [4] for a review and [5], [6], [7] for specific techniques aimed at manipulation planning. These methods generally provide probabilistic completeness, but suffer from the limitation that configuration space samples are connected by local paths lying outside the constrained manifold. To improve task tracking accuracy, it is necessary to use a more dense sampling, typically leading to a dramatic increase of the time needed to compute a plan. For complex problems, this approach can turn out to be very inefficient and impractical.

In [8], we introduced a sampling-based approach for solving the TCMP problem that avoids the need for a projection mechanism thanks to a control-based motion generation scheme; as a consequence, it becomes possible to guarantee continuous satisfaction of the task constraint with arbitrary precision. Building on this basic technique, we have proposed a method for repeatable motion planning over cyclic tasks [9] and another for planning dynamically feasible motions in the presence of moving obstacles [10]. All these

planners are designed for the case of hard constraints.

Most motion planners for the case of soft task constraints rely on some sort of relaxation of the equality constraint represented by the assigned task path or trajectory [11], [12]. In [13], [14], planning is performed over an approximation of the constrained configuration space.

All the above methods attempt to satisfy only the soft task constraint throughout the motion. We argue, however, that it would be interesting (and presumably practical) to develop an opportunistic planner which is capable of satisfying the hard constraint whenever possible and of exploiting the available tolerance only when needed. In particular, deviations from the given task path should only take place in the presence of obstructions in the constrained configuration space, such as a narrow or closed passage. Narrow passages are a well-known problematic issue for sampling-based planners; among the most successful methods to handle them, one may mention Gaussian sampling [15] and the bridge test [16]; see [17], [18] for reviews on the topic. Very few works look at narrow passages in a task-constrained setting; one example is [19].

The objective of this paper is precisely to present a motion planner for the case of soft task constraints that is, for the first time, opportunistic in the sense defined above. To this end, we provide the following contributions:

- a hard planner for generating collision-free motions that realize exactly the desired task path, essentially an adaptation of our control-based randomized method for TCMP [8];
- a heuristic criterion to detect obstructions to the hard planner due to narrow/closed passages in the constrained configuration space;
- a soft planner for generating collision free-motions that are compliant with the soft task and allow to bypass the obstructions detected by the hard planner;
- another heuristic criterion to estimate when the obstructions to hard planning have been removed.

The paper is organized as follows. Section II provides a precise formulation of the considered planning problem. An overview of the proposed opportunistic planner is given in Sect. III, while the hard and soft subplanners are described in Sects. IV and V, respectively. Several planning experiments for the PR2 mobile manipulator are shown in Sect. VI. Finally, in Sect. VII we discuss some possible future developments.

II. Problem Formulation

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

Assume that the robot is free-flying (i.e., its configuration can move arbitrarily in $\mathcal{C}$), so that its kinematic model consists of simple integrators. In order to plan paths, we use a geometric version [10] of such model, expressed as $\boldsymbol{q}' = \tilde{\boldsymbol{v}}$, where $()' = d()/ds$ denotes the derivative w.r.t. the path parameter s. This equation entails that the tangent vectors to any path $\boldsymbol{q}(s)$ in $\mathcal{C}$ can be chosen arbitrarily by specifying the (geometric) inputs $\tilde{\boldsymbol{v}}$.

The task is described in coordinates $\boldsymbol{t}$, taking values in an n_t-dimensional task space $\mathcal{T}$. Coordinates $\boldsymbol{t}$ and $\boldsymbol{q}$ are related by the forward kinematic map $\boldsymbol{t} = \boldsymbol{f}(\boldsymbol{q})$, which at the tangent vector level becomes $\boldsymbol{t}' = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{q}'$, where $\boldsymbol{J}(\boldsymbol{q}) = d\boldsymbol{f}/d\boldsymbol{q}$ is the task Jacobian. We will assume that $n_q > n_t$, i.e., the robot is kinematically redundant for the assigned task.

In the situation of interest, shown in Fig. 1, the robot is assigned a *soft task* defined by

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

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

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

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

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

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

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

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

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

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

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

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

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

[1] This inequality and similar ones in the paper are meant componentwise.

III. Overview of the Opportunistic Planner

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

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

The pseudocode of the proposed planner is given in Algorithm 1. Construction of T starts by rooting it at the initial $\boldsymbol{q}_{\text{ini}}$. At the generic iteration, let h ($h = 0, \ldots, N-1$) be the *frontier index*, i.e., the index of the largest sample of s for which there exists a vertex $\boldsymbol{q}$ in T on $\mathcal{L}_h$ or $\mathcal{S}_h$.

First, HP is invoked in the attempt to extend T as much as possible in $\mathcal{C}_{\text{hard}}$. When HP stops, it returns an updated value of h. If $h < N$, it means that HP has heuristically identified an obstruction to extending T from $\mathcal{L}_h$ through subpaths in $\mathcal{C}_{\text{hard}}$; this may indicate that the extension is simply difficult, due to the presence of a narrow passage in $\mathcal{C}_{\text{hard}} \cap \mathcal{C}_{\text{free}}$, or actually impossible, as it happens when an obstacle occupies a portion of the desired task path. At this point, SP is invoked to allow extension of T from $\mathcal{L}_h$ in $\mathcal{C}_{\text{soft}}$, the rationale being that this may overcome the obstruction. When SP stops, it also returns an updated current value of h, now representing the index associated to the value of s where extension by HP is considered viable again; HP is invoked, and the procedure is repeated (see Fig. 2). SP returns $h = \emptyset$ if it does not succeed in extending T from $\mathcal{L}_h$. The inner loop (lines 3-7 of Algorithm 1) continues until $h = N$ or $h = \emptyset$. In the first case, a configuration $\boldsymbol{q}_{\text{fin}}$ exists in T such that $\boldsymbol{q}_{\text{fin}} \in \mathcal{L}_N$ or $\boldsymbol{q}_{\text{fin}} \in \mathcal{S}_N$ (depending on whether it has been generated by HP or SP), and a path $\overline{\boldsymbol{q}_{\text{ini}}\boldsymbol{q}_{\text{fin}}}$ can be extracted from T. In the second case, the planner returns a failure.

A remark is in order here about the choice of N, i.e., the number of subsets $\mathcal{L}_i$ (or $\mathcal{S}_i$) used by our planner. While

Algorithm 1: Opportunistic Planner

```
1  T.AddVertex(q_ini);
2  h ← 0;
3  repeat
4  |   h ← HP(T, h);
5  |   if h < N then
6  |   |   h ← SP(T, h);
7  until h = ∅ or h = N;
8  if h = N then
9  |   q_ini q_fin ← T.RetrievePath();
10 |   return q_ini q_fin;
11 return ∅;
```

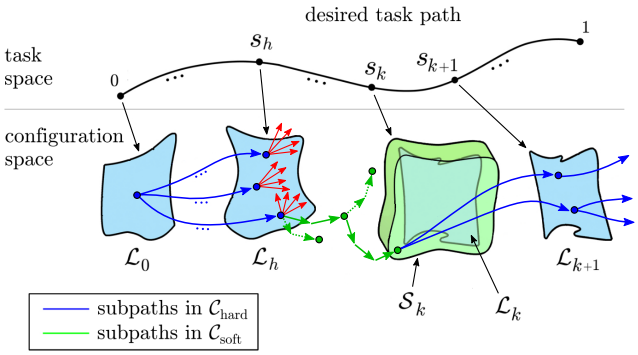

Fig. 2. The tree T built by the opportunistic planner (solid blue/green). The portion of T up to $\mathcal{L}_h$ (solid blue) has been generated by HP and is contained in $\mathcal{C}_{\text{hard}}$. The red arrows originating at vertexes on $\mathcal{L}_h$ indicate a number of failed extension attempts, after which SP has been invoked to extend T in $\mathcal{C}_{\text{soft}}$. To this end, SP identifies the first task sample s_k where the obstruction has disappeared, and grows an auxiliary tree T_{soft} (dashed green) towards $\mathcal{S}_k$. When T_{soft} has reached $\mathcal{S}_k$, the subpath from $\mathcal{L}_h$ to $\mathcal{S}_k$ (solid green) is added as an edge to T. At this point, control goes back to HP, and extension of T is in $\mathcal{C}_{\text{hard}}$ is resumed (solid blue).

N has no impact on the accuracy with which the desired task path is realized, it is true that larger values of N allow a finer generation of subpaths in T, ultimately increasing the possibility of navigating the robot among obstacles. However, the size of the tree will grow accordingly, for vertexes will have to be placed on a larger number of subsets $\mathcal{L}_i$ (by HP) or $\mathcal{S}_i$ (by SP). The value of N must therefore represent a reasonable trade-off between maneuvrability of the robot and complexity of planning.

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

IV. Hard Planner

HP is essentially an adaptation of the control-based planner proposed in [8], with the addition of a heuristic-based mechanism for detecting obstructions to further tree extension in $\mathcal{C}_{\text{hard}}$. The pseudocode of HP is given in Algorithm 2.

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

$$\tilde{\boldsymbol{v}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})(\boldsymbol{t}'_d + k_t \boldsymbol{e}_t(\boldsymbol{q})) + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}))\tilde{\boldsymbol{w}}, \qquad (2)$$

where $\boldsymbol{J}^{\dagger}$ is the pseudoinverse of the task Jacobian $\boldsymbol{J}$, k_t is a positive gain, $\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J}$ is the orthogonal projection matrix in the null space of $\boldsymbol{J}$, and $\tilde{\boldsymbol{w}}$ is a n_q-dimensional vector that represents a *residual* geometric input (it can be chosen arbitrarily without affecting task execution). To allow effective exploration of the planning space, $\tilde{\boldsymbol{w}}$ is randomly generated with bounded norm. Subpaths generated from $\mathcal{C}_{\text{hard}}$ via (2) are guaranteed to remain in $\mathcal{C}_{\text{hard}}$; whereas they converge exponentially to $\mathcal{C}_{\text{hard}}$ if $\boldsymbol{q}_{\text{near}}$ is outside it.

Algorithm 2: HP(T, h)

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

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

The above extension procedure is iterated until the frontier index h reaches N, or an obstruction is detected to further extension of T. As explained in the previous section, the latter may be due to a narrow or even closed passage in $\mathcal{C}_{\text{hard}} \cap \mathcal{C}_{\text{free}}$. To identify such situations, a heuristic criterion is used. In particular, HP will assume that an obstruction exists if both the following conditions are satisfied (see Fig. 3):

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

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

V. Soft Planner

SP, whose pseudocode is given in Algorithm 3, is invoked when HP detects an obstruction to further extension of T in $\mathcal{C}_{\text{hard}}$ from $\mathcal{L}_h$. SP first identifies the index k ($k = h+1, \ldots, N$) associated to the first task sample s_k where the obstruction disappears, and then grows an auxiliary tree T_{soft} in $\mathcal{C}_{\text{soft}}$ to connect $\mathcal{L}_h$ to $\mathcal{S}_k$. Once a connecting subpath has been found, it is extracted from T_{soft} and added as an edge to T. At this point, HP resumes planning in $\mathcal{C}_{\text{hard}}$ (Fig. 2).

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

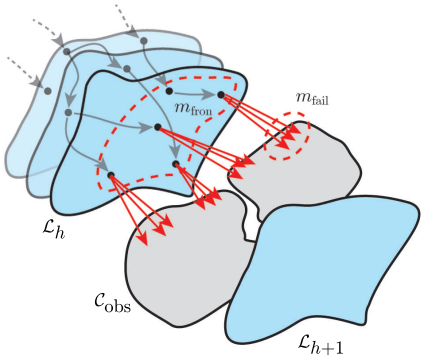

Fig. 3. The proposed heuristic criterion detects an obstruction if (1) the number m_{fron} of vertexes on the frontier leaf has reached a threshold $m_{\text{fron}}^{\max}$, and (2) the number $m_{\text{fail}}(\boldsymbol{q}_j)$ of failed extension attempts from each such vertex $\boldsymbol{q}_j$ has reached a threshold $m_{\text{fail}}^{\max}$.

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

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

- $\boldsymbol{q}_{\text{new}} \in \mathcal{S}_k$, i.e., $\boldsymbol{q}_{\text{new}}$ is compliant with the soft task at s_k. In this case, the subpath $\overline{\boldsymbol{q}_h\boldsymbol{q}_{\text{new}}}$ is extracted from T_{soft} and added to T together with $\boldsymbol{q}_{\text{new}}$ as, respectively, a new edge and a new vertex.
- A maximum number $i_{\max}$ of extension attempts is exceeded. In this case, a failure is reported.

A. Soft Tree Extension

The SP extension procedure, whose pseudocode is given in Algorithm 4, generates a subpath $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ in $\mathcal{C}_{\text{soft}}$ through a sequence of configurations that are valid and compliant with the soft task for increasing values of s. To this end, it uses a fine discretization of the $[s_h, s_k]$ interval with a step δs, producing the equispaced sequence $\{s_h, s_h+\delta s, \ldots, s_h+M\cdot\delta s = s_k\}$, with $M = (s_k - s_h)/\delta s$. Every new configuration generated during extension will be associated to a unique value of s within the above sequence.

Extension begins by taking a step of length η from $\boldsymbol{q}_{\text{near}}$ towards a random configuration $\boldsymbol{q}_{\text{rand}}$; let $\boldsymbol{q}_{\text{curr}}$ be the generated configuration and s_{curr} the associated parameter

Algorithm 3: SP(T, h)

$k \leftarrow h$;
repeat
  $k \leftarrow k + 1$;
  GenerateIKSolutions(m_{sol}, $\mathcal{L}_k$);
  $m_{\text{free}} \leftarrow$ CountCollisionFreeSolutions();
until $m_{\text{free}} \geq m_{\text{free}}^{\min}$ **or** $k = N$;
$\boldsymbol{q}_h \leftarrow$ RandomVertex($T, \mathcal{L}_h$);
T_{soft}.AddVertex($\boldsymbol{q}_h$);
$i \leftarrow 0$;
repeat
  $\boldsymbol{q}_{\text{rand}} \leftarrow$ RandomConfig($\mathcal{C}_{\text{free}}$);
  $\boldsymbol{q}_{\text{near}} \leftarrow$ NearestVertex($T_{\text{soft}}, \boldsymbol{q}_{\text{rand}}$);
  $(\boldsymbol{q}_{\text{new}}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}) \leftarrow$ ExtendSoft($\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}}, k$);
  if $\boldsymbol{q}_{\text{new}} \neq \emptyset$ **then**
    T_{soft}.Add($\boldsymbol{q}_{\text{new}}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$);
  $i \leftarrow i + 1$;
until $\boldsymbol{q}_{\text{new}} \in \mathcal{S}_k$ **or** $i = i_{\max}$;
if $\boldsymbol{q}_{\text{new}} \in \mathcal{S}_k$ **then**
  $\overline{\boldsymbol{q}_h\boldsymbol{q}_{\text{new}}} \leftarrow T_{\text{soft}}$.RetrievePath();
  T.Add($\boldsymbol{q}_{\text{new}}, \overline{\boldsymbol{q}_h\boldsymbol{q}_{\text{new}}}$);
  return k;
return $\emptyset$;

Algorithm 4: ExtendSoft($\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}}, k$)

$\boldsymbol{q}_{\text{curr}} \leftarrow \boldsymbol{q}_{\text{near}} + \eta \frac{\boldsymbol{q}_{\text{rand}} - \boldsymbol{q}_{\text{near}}}{\|\boldsymbol{q}_{\text{rand}} - \boldsymbol{q}_{\text{near}}\|}$;
compute s_{curr} (3);
repeat
  compute $\boldsymbol{e}_t(\boldsymbol{q}_{\text{curr}}, s_{\text{curr}} + \delta s)$(4);
  compute $\boldsymbol{d}$ (5);
  $\boldsymbol{q}_{\text{curr}} \leftarrow \boldsymbol{q}_{\text{curr}} + \eta \boldsymbol{d}$;
  compute s_{curr} (3);
until $s_{\text{curr}} = s_k$ **or** $s_{\text{curr}} = \emptyset$ **or** !Valid($\boldsymbol{q}_{\text{curr}}$);
if $s_{\text{curr}} = s_k$ **then**
  return $[\boldsymbol{q}_{\text{curr}}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{curr}}}]$;
return $[\boldsymbol{q}_{\text{curr}}$.parent, $\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{curr}}$.parent$]$;

sample, i.e., the smallest value in the subsequence $\{s_{\text{near}} + \delta s, \ldots, s_k\}$ for which $\boldsymbol{q}_{\text{curr}}$ is compliant with the soft task:

$$s_{\text{curr}} = \min s \in \{s_{\text{near}} + \delta s, \ldots, s_k\} : \boldsymbol{q}_{\text{curr}} \in \mathcal{S}(s), \quad (3)$$

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

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

$$\boldsymbol{e}_t(\boldsymbol{q}_{\text{curr}}, s_{\text{curr}} + \delta s) = \boldsymbol{t}_d(s_{\text{curr}} + \delta s) - \boldsymbol{f}(\boldsymbol{q}_{\text{curr}}). \quad (4)$$

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

$$\boldsymbol{d} = -\frac{\boldsymbol{J}^T(\boldsymbol{q}_{\text{curr}})\boldsymbol{e}_t(\boldsymbol{q}_{\text{curr}}, s_{\text{curr}} + \delta s)}{\left\|\boldsymbol{J}^T(\boldsymbol{q}_{\text{curr}})\boldsymbol{e}_t(\boldsymbol{q}_{\text{curr}}, s_{\text{curr}} + \delta s)\right\|}, \quad (5)$$

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

The cycle is interrupted in the following cases:

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

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

VI. Planning Experiments

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

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

The same parameters are used in all scenarios:

- the samples of the desired task path are $N + 1 = 11$;
- $k_t = 10$ in the motion generation (2), which is integrated with Euler method and a stepsize of 0.002;
- HP detects the presence of a planning obstruction by setting $m_{\text{fron}}^{\max} = m_{\text{fail}}^{\max} = 5$, while SP identifies its absence using $m_{\text{sol}} = 100$ and $m_{\text{free}}^{\min} = 20$;
- SP extension works with $\eta = 0.01$ and $\delta s = 0.02$.

The tolerance is specified in the local frame[3] of the desired task path, as this is the most simple and intuitive option for the user. Accordingly, compliance with the soft task at a certain configuration is evaluated using (1) by expressing the task error in that frame.

For each scenario, we report some snapshots from a solution (Figs. 4–7) as well as the evolution of the task error along that solution (Fig. 8). To better appreciate the quality of the generated motions, we invite the reader to watch the accompanying video, which shows animated clips of the solutions.

In the first scenario (Fig. 4), the desired task path for the end-effector is a line passing through a pillar. The tolerance is specified as $\Delta\boldsymbol{t} = (0.07, 0.2, 0.1)$ m (corresponding to the green volume in Fig. 1). In the first part of the motion, HP is able to execute the desired task path (snapshots 1 and 2). In the vicinity of the pillar, HP detects an obstruction and SP is invoked; this leads to an end-effector path that

[3]In our implementation, such frame has the origin at $\boldsymbol{t}_d(s)$ and the x- and y-axes oriented, respectively, as $\boldsymbol{t}'_d(s) = (x'_d, y'_d, z'_d)$ and $(y'_d, -x'_d, 0)$; the z-axis is consequently defined. With this choice, the x-axis is always tangent to $\boldsymbol{t}_d(s)$ and oriented along the direction induced on $\boldsymbol{t}_d(s)$ by s.

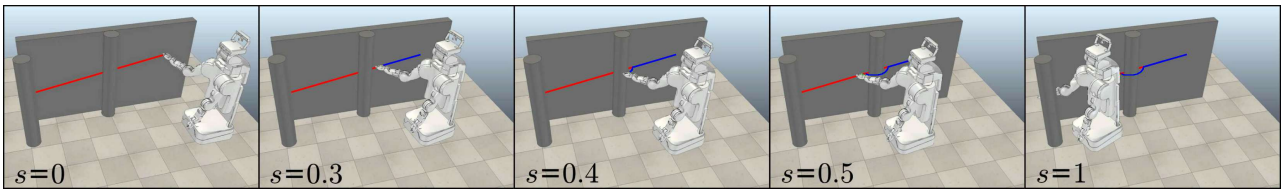

Fig. 4. Planning scenario 1: snapshots from a solution. The desired and actual task paths are shown in red and blue, respectively. The robot leaves the desired path only when strictly necessary to avoid the pillar.

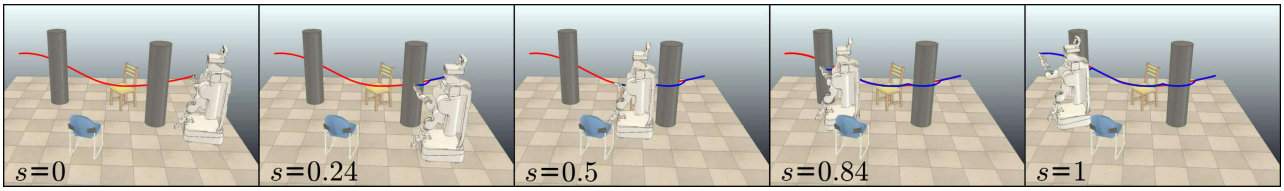

Fig. 5. Planning scenario 2: snapshots from a solution. The task tolerance is exploited in correspondence of the two portions of the desired path that are obstructed by pillars.

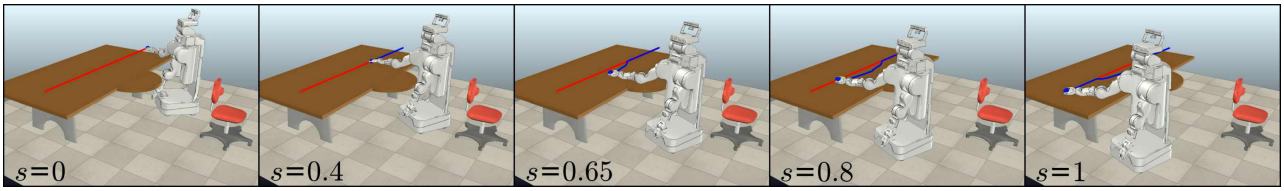

Fig. 6. Planning scenario 3: snapshots from a solution. The robot carries an object from a location to another above the table, leaving the desired end-effector path only when strictly necessary to avoid collisions between its torso and the table.

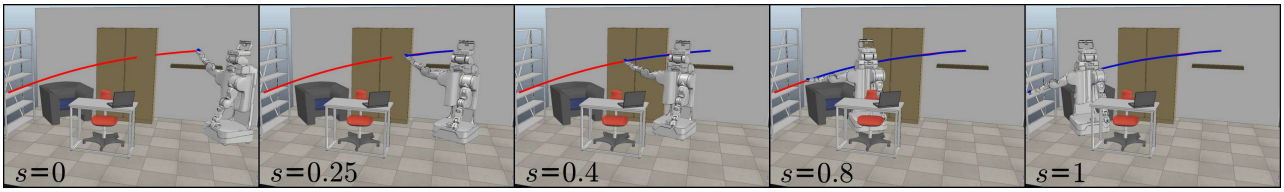

Fig. 7. Planning scenario 4: snapshots from a solution. Although the environment is quite cluttered, the robot succeeds in carrying the object along the desired path, momentarily deviating from it only in order to avoid the cabinet.

deviates (snapshot 3) from the desired one for $s \in [0.3, 0.5]$, still remaining inside the available tolerance (Fig. 8). As soon as SP considers the obstruction to have disappeared, HP takes back control and the robot returns on the desired path (snapshot 5).

The second scenario (Fig. 5) is aimed at confirming that the opportunistic planner is able to leave and return to the desired task path multiple times. To this end, the robot end-effector is assigned a sinusoidal path that passes through two pillars, with the tolerance defined as $\Delta \boldsymbol{t} = (0.07, 0.3, 0.1)$ m. Snapshots 2 and 4 show that the robot correctly exploits the tolerance twice, for $s \in [0.1, 0.3]$ and $s \in [0.7, 0.9]$, while the desired path is realized everywhere else (see also Fig. 8).

In the third scenario (Fig. 6) the robot must execute a simple pick and place task, i.e., moving a ball from an initial to a final position on the table. In such a situation, the user may specify a very simple (and time-efficient) tentative task path, defined as the line segment joining the two locations above the pick and place positions. This path may be abandoned if necessary, but for safety reasons it is desirable that the object always remains above (but not in contact with) the table during the motion. Such requirements translate to a tolerance specified as $\Delta \boldsymbol{t} = (0.07, 0.3, 0.1)$ m. The results show that the desired path is perfectly executed at the start and at the end, with the robot retracting its end-effector for $s \in [0.4, 0.8]$ (snapshots 2-4) to avoid collision between its torso and the table. As indicated by Fig. 8, the task error is always within the tolerance region.

The final scenario (Fig. 7) also deals with a pick and place task, with the robot now required to move the ball from a shelf to a desired location on a bookcase. The task is specified through a curved desired path and a tolerance $\Delta \boldsymbol{t} = (0.05, 0.25, 0.1)$ m. An early portion of the desired path goes through a cabinet, while the second part requires the robot to navigate a very cluttered region. As in previous scenarios, the desired path is initially realized (snapshot 1) and, under the action of SP, briefly abandoned for $s \in [0.1, 0.4]$ in order to avoid the cabinet (snapshots 2 and 3). Once such obstruction has been removed, HP takes back control and brings back the robot to the desired path, staying on it until the end in

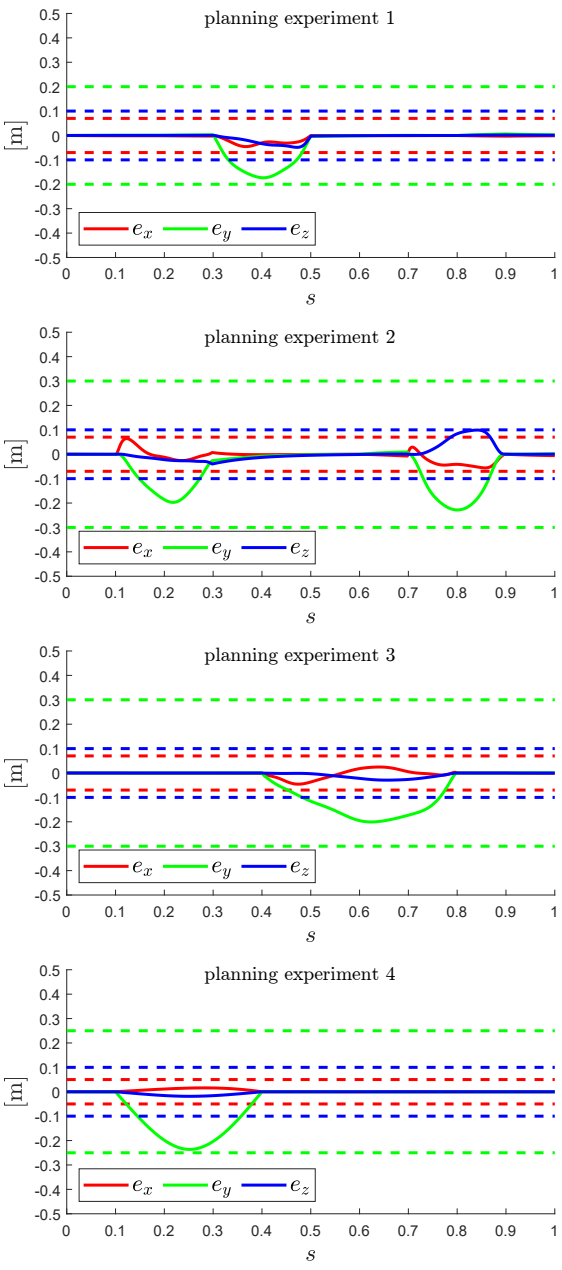

Fig. 8. Evolution of the task error $\boldsymbol{e}_t$ in the four planning experiments. The dashed lines indicate the tolerance for each component.

spite of the very limited workspace clearance.

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

VII. Conclusions

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

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

TABLE I
AVERAGED PERFORMANCE DATA.

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

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

References

[1] S. Chiaverini, G. Oriolo, and A. A. Maciejewski, "Redundant robots," in *Springer Handbook of Robotics – 2nd Edition*, B. Siciliano and O. Khatib, Eds. Springer-Verlag Berlin Heidelberg, 2016, pp. 221–242.

[2] 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.

[3] D. Rakita, B. Mutlu, and M. Gleicher, "Stampede: A discrete-optimization method for solving pathwise-inverse kinematics," in *2019 IEEE Int. Conf. on Robotics and Automation*, 2019, pp. 3507–3513.

[4] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 159–185, 2018.

[5] G. Oriolo and C. Mongillo, "Motion planning for mobile manipulators along given end-effector paths," in *2005 IEEE Int. Conf. on Robotics and Automation*, 2005, pp. 2166–2172.

[6] M. Stilman, "Task constrained motion planning in robot joint space," in *2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007, pp. 3074–3081.

[7] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *Int. J. of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[8] 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*, 2009, pp. 297–302.

[9] G. Oriolo, M. Cefalo, and M. Vendittelli, "Repeatable motion planning for redundant robots over cyclic tasks," *IEEE Trans. on Robotics*, vol. 33, no. 5, pp. 1170–1183, 2017.

[10] M. Cefalo and G. Oriolo, "A general framework for task-constrained motion planning with moving obstacles," *Robotica*, vol. 37, pp. 575–598, 2019.

[11] T. Kunz and M. Stilman, "Manipulation planning with soft task constraints," in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012, pp. 1937–1942.

[12] M. Guo and M. M. Zavlanos, "Probabilistic motion planning under temporal tasks and soft constraints," *IEEE Trans. on Automatic Control*, vol. 63, no. 12, pp. 4051–4066, 2018.

[13] I. Sucan and S. Chitta, "Motion planning with constraints using configuration space approximations," in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012, pp. 1904–1910.

[14] Z. Fusheng, L. Yizhou, G. Wei, W. Pengfei, L. Mantian, W. Xin, and L. Jingxuan, "Learning the metric of task constraint manifolds for constrained motion planning," *Electronics*, vol. 7, p. 395, 2018.

[15] V. Boor, M. Overmars, and A. V. der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *1999 IEEE Int. Conf. on Robotics and Automation*, 1999, pp. 1018–1023.

[16] D. Hsu, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *2003 IEEE Int. Conf. on Robotics and Automation*, 2003, pp. 4420–4426.

[17] Z. Sadeghi and H. Moradi, "A new sample-based strategy for narrow passage detection," in *2011 9th IEEE World Congress on Intelligent Control and Automation*, 2011, pp. 1059–1064.

[18] M. Saha, J.-C. Latombe, Y.-C. Chang, and F. Prinz, "Finding narrow passages with probabilistic roadmaps: The small-step retraction method," *Autonomous Robots*, vol. 19, no. 3, pp. 301–319, 2005.

[19] D. Berenson, T. Siméon, and S. Srinivasa, "Addressing cost-space chasms in manipulation planning," in *2011 IEEE Int. Conf. on Robotics and Automation*, 2011, pp. 4561–4568.