Robotica (2019) volume 37, pp. 575–598. © Cambridge University Press 2018
doi:10.1017/S0263574718001182

A general framework for task-constrained motion planning with moving obstacles

Massimo Cefalo and Giuseppe Oriolo*

Dipartimento di Ingegneria Informatica, Automatica e Gestionale,
Sapienza Università di Roma, Via Ariosto 25, Rome 00185, Italy. E-mail: cefalo@diag.uniroma1.it

(Accepted October 8, 2018. First published online: October 30, 2018)

SUMMARY

Consider the practically relevant situation in which a robotic system is assigned a task to be executed in an environment that contains moving obstacles. Generating collision-free motions that allow the robot to execute the task while complying with its control input limitations is a challenging problem, whose solution must be sought in the robot state space extended with time. We describe a general planning framework which can be tailored to robots described by either kinematic or dynamic models. The main component is a control-based scheme for producing configuration space subtrajectories along which the task constraint is continuously satisfied. The geometric motion and time history along each subtrajectory are generated separately in order to guarantee feasibility of the latter and at the same time make the scheme intrinsically more flexible. A randomized algorithm then explores the search space by repeatedly invoking the motion generation scheme and checking the produced subtrajectories for collisions. The proposed framework is shown to provide a probabilistically complete planner both in the kinematic and the dynamic case. Modified versions of the planners based on the exploration–exploitation approach are also devised to improve search efficiency or optimize a performance criterion along the solution. We present results in various scenarios involving non-holonomic mobile robots and fixed-based manipulators to show the performance of the planner.

KEYWORDS: Motion planning; Task constraints; Moving obstacles; Randomized planners; Redundant robots

1. Introduction

Motion planning among obstacles is a classical problem in robotics. Two main versions of the problem can be considered, depending on whether the obstacles are fixed or moving. In particular, the latter version is attracting more and more interest as robots are increasingly called to share their workspace with humans or other robots. This is obviously true in service robotics, where robots must operate in everyday life contexts, but also in factory automation, with the paradigm of human–robot collaboration becoming widely adopted.

Motion planning among fixed obstacles is essentially a geometric problem, for which any path contained in the free configuration space represents a solution. Removing the assumption that obstacles are fixed leads to a problem which is no longer purely geometric. To generate collision-free motions in the presence of moving obstacles, in fact, it is necessary to plan an appropriate time history in conjunction with the configuration space path. As a consequence, motion planning among moving obstacles proves to be a very challenging problem, even if one assumes that the trajectories of the obstacles are completely known in advance. This was first shown in ref. [1] for a single-body robot which can move at arbitrary velocity. Most early works were based on the idea of modifying existing combinatorial or sampling-based methods for searching the configuration space extended with the time dimension, see, e.g., refs. [2–4]. Another interesting approach was based on the so-called velocity obstacle technique, see refs. [5–7]. In ref. [8], where the expression *kinodynamic planning* was coined, the planning problem among moving obstacles was considered for a point mass robot subject to

* Corresponding author. E-mail: oriolo@diag.uniroma1.it

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

acceleration constraints. The sampling-based algorithm proposed in ref. [9] for the case of moving obstacles considers both kinematic and dynamic constraints. Although conceptually relevant, none of the above methods is suitable for planning in complex, high-dimensional configuration spaces due to their computational complexity.

The hypothesis that the trajectories are known in advance may be relaxed by introducing uncertainty in the problem through the use of probabilistic models,[10] or even completely discarded by assuming no previous knowledge at all on the obstacle motion.[11] In these contexts, one typically resorts to on-line motion planning based on sensor measurements. Planning approaches range from purely reactive strategies, which suffer however from local minima and suboptimal performance, to hybrid techniques that combine a global planner with local reactive behaviors (see, e.g., ref. [12]). Another family of methods that can be used in dynamic environments are anytime planners.[13,14]

In this work, we shall keep the hypothesis that the obstacles move along known trajectories, and therefore look at the motion planning problem from an off-line point of view. This is a rather strong assumption that is however satisfied in many situations of interest, especially in the factory automation context. In any case, we argue that solving the problem with complete information represents the first step toward the development of on-line planners that can work in the presence of partial information.

The methods for motion planning discussed so far cannot incorporate the task constraints which arise in many applications. In industrial settings, manipulators are often required to move their end effector along predefined paths, e.g., for welding, drawing, cutting, or assembling. For mobile robots used in service applications, tasks can be of a more general kind, e.g., navigating a corridor, tracking visual features, carrying an object, maintaining mechanical balance, and so on. Constrained motion planning, or more precisely, task-constrained motion planning (henceforth TCMP), is therefore acknowledged as a practically relevant extension of the motion planning problem.

A common approach to devise TCMP planners is to use a randomized search algorithm such as PRM[15] or RRT,[16] and then project the samples on the admissible submanifold of the configuration space where the task constraint is satisfied. The projection mechanism can be realized via randomized gradient descent, tangent space sampling, or retraction, e.g., see refs. [17–20].

In ref. [21], we introduced an innovative method for TCMP which avoids altogether the need for projection by using a control-based mechanism for constrained motion generation. As a consequence, this planner is able to guarantee continuous satisfaction of the task constraint at arbitrary precision without increasing the time needed to find a solution. The same control-based approach was then extended to TCMP with moving obstacles (henceforth TCMP_MO) in refs. [22] and [23], for robots described, respectively, by kinematic and dynamic models.

This paper fully develops the ideas contained in our previous papers into a complete framework for TCMP in the presence of obstacles. In particular, it adds the following contributions:

- A unified planning approach which is readily tailored to robots described by kinematic or dynamic models.
- A technique for satisfying the bounds on the control inputs by construction rather than by validation.
- An extension to kinematic robots subject to non-holonomic constraints.
- A mechanism for balancing exploration and exploitation during the search in order to improve its efficiency or optimize a performance criterion.
- A proof of probabilistic completeness.
- Numerical results in new complex scenarios, including the case of a multi-robot system.

The paper is organized as follows. In the next section, we discuss the fundamental aspects of the considered planning problem. Section 3 addresses the case of robots described by kinematic models: in particular, Section 3.1 provides the specific formulation of the problem, Section 3.2 discusses the nature of the associated search space, Section 3.3 describes the proposed planning algorithm, and Section 3.4 considers the extension to non-holonomic robots. Section 4 follows exactly the same structure with reference to the case of robots described by dynamic models. The probabilistic completeness of both planners is established in Section 5, and their modification in an exploration–exploitation sense is outlined in Section 6. Finally, Section 7 presents numerical results obtained in scenarios involving mobile robots as well as fixed-base manipulators. A pseudocode description of the proposed planners is given in the Appendix.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

2. Core Problem

Consider a robotic system whose configuration $\boldsymbol{q}$ (the vector of generalized coordinates) takes values in an n-dimensional configuration space $\mathcal{C}$. The system moves in a three-dimensional workspace $\mathcal{W}$ populated by moving obstacles. Denote by $\mathcal{R}(\boldsymbol{q}) \subset \mathcal{W}$ the volume occupied by the robot at configuration $\boldsymbol{q}$, and by $\mathcal{O}(t) \subset \mathcal{W}$ the volume occupied by the obstacles at time t.

The robot is assigned a task expressed in coordinates $\boldsymbol{y}$, which take values in an m-dimensional task space $\mathcal{Y}$. Typical task coordinates 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). In any case, they are related to the configuration coordinates by a forward kinematic map:

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{q}). \tag{1}$$

In particular, the task is assigned as a desired *path* $\boldsymbol{y}_d(s)$ for the $\boldsymbol{y}$ coordinates, with the path parameter s (e.g., the arc length) taking values in $[s_{\text{ini}}, s_{\text{fin}}]$.

The TCMP_MO (Task-Constrained Motion Planning with Moving Obstacles) problem consists in searching for a configuration space *trajectory* $\boldsymbol{q}(t)$ of the robot such that (1) the trajectory itself is feasible for the considered robot, (2) the assigned task path is exactly executed, and (3) all collisions between the robot and the obstacles are avoided.

We emphasize the following points:

- The above formulation, with assigned task *path* and known obstacle *trajectories*, is the only instance of TCMP_MO that represents an open problem. In fact, if a task *trajectory* is assigned, one can directly use the method of ref. [21], with the simple adaptation of checking collisions with respect to the moving obstacles.
- Each solution $\boldsymbol{q}(t)$ of the TCMP_MO problem actually consists of two components: a configuration space path $\boldsymbol{q}(s) : [s_{\text{ini}}, s_{\text{fin}}] \mapsto \mathcal{C}$, and a time history $s(t) : [0, T] \mapsto [s_{\text{ini}}, s_{\text{fin}}]$ along the path.
- The time history $s(t)$ is not required to be monotonic: s must start at s_{ini} and end at s_{fin}, but at any point along the path s may increase or decrease. This possibility can be exploited by the planner in order to avoid moving obstacles more effectively.
- Both the duration T of the motion and the final configuration $\boldsymbol{q}(s_{\text{fin}}) = \boldsymbol{q}(T)$ are not specified in advance and will be a byproduct of the solution.

We shall take the following assumptions.

A1: The robot is kinematically redundant for the task, i.e., $n > m$ (the number of generalized coordinates exceeds the task dimension).

A2: We have complete *a priori* information on the obstacle motion, i.e., $\mathcal{O}(t) \subset \mathcal{W}$ is known for all t.

A3: The initial robot configuration $\boldsymbol{q}(0) = \boldsymbol{q}_{\text{ini}}$ is assigned.

Assumption A1 is necessary for the planning problem to be well posed; in the absence of redundancy, in fact, there would be a one-to-one mapping between paths in $\mathcal{C}$ and in $\mathcal{Y}$, and therefore the TCMP_MO problem would degenerate to a simple time scaling problem along the task path.

Assumption A2 sets us in an *off-line* planning context. This may be more realistic in industrial settings than, e.g., in service robotics. In any case, solving the problem with complete information represents the first step toward the development of an on-line planner that can work in the presence of partial information.

As for Assumption A3, if $\boldsymbol{q}_{\text{ini}}$ is not actually specified, it can be easily generated via inverse kinematics as any robot configuration which realizes the initial task value $\boldsymbol{y}_d(0)$.

Depending on the adopted robot model, and the bounds on the associated control inputs, we will obtain two versions of the TCMP_MO problem: a kinematic version and a dynamic version. In the following sections, we discuss in detail each problem, highlighting their fundamental similarities and proposing a planning approach that can be easily tailored for both contexts.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

3. The Kinematic Case

In the first version of the problem, we assume that the robot is modeled as a fully actuated, purely kinematic system which is free of constraints[1]. Correspondingly, the input bounds concern the generalized velocities. This is an appropriate description for robots whose motion is well reflected in a free-flying kinematic model (e.g., omnidirectional mobile robots) or for digital animation purposes. In the last part of this section, we will extend the proposed planner to kinematic systems subject to non-holonomic constraints.

3.1. Problem formulation

Assume that the robot is described by the following kinematic-level model:

$$\dot{\boldsymbol{q}} = \boldsymbol{v}, \tag{2}$$

where $\boldsymbol{v} = (v_1, \ldots, v_n)$ represents the vector of generalized velocity inputs. These are subject to bounds of the form

$$|v_i| \leq v_i^M, \quad i = 1, \ldots, n \qquad \text{or in compact form} \qquad |\boldsymbol{v}| \leq \boldsymbol{v}^M, \tag{3}$$

which are relevant for the TCMP_MO problem because they ultimately limit the speed (positive or negative) at which the robot can execute the task path. A trajectory in $\mathcal{C}$ that satisfies these bounds is called *feasible*.

Under assumptions A1–A3, a solution to the *kinematic version* of the TCMP_MO problem (called K_TCMP_MO in the following) consists of a path $\boldsymbol{q}(s) : [s_{\text{ini}}, s_{\text{fin}}] \mapsto \mathcal{C}$ and a continuous time history $s(t) : [0, T] \mapsto [s_{\text{ini}}, s_{\text{fin}}]$ such that the following conditions are satisfied:

C1: $|\dot{\boldsymbol{q}}| \leq \boldsymbol{v}^M$, for $t \in [0, T]$.
C2: $s(0) = s_{\text{ini}}$ and $s(T) = s_{\text{fin}}$.
C3: $\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{q}(s(t))) = \boldsymbol{y}_d(s(t))$ for all $t \in [0, T]$.
C4: $\boldsymbol{q}(s_{\text{ini}}) = \boldsymbol{q}_{\text{ini}}$.
C5: $\mathcal{R}(\boldsymbol{q}(s(t))) \cap \mathcal{O}(t) = \emptyset$, for all $t \in [0, T]$.

Condition C1 implies that the generated motion is feasible. C2 requires the robot to start at the beginning and to stop at the end of the task path; in view of the continuity of $s(t)$, it also guarantees that all values of s in $[s_{\text{ini}}, s_{\text{fin}}]$ are generated by the time history. C3 ensures that the task path will be entirely and exactly realized. C4 entails that the robot starts at the assigned initial configuration $\boldsymbol{q}_{\text{ini}}$. Finally, C5 guarantees that collisions with all the obstacles are avoided.[2]

Note that the generalized velocities involved in Condition 1 can be expressed as

$$\dot{\boldsymbol{q}} = \dot{s}\, \boldsymbol{q}', \tag{4}$$

where we have used the notation $(\,)' = d(\,)/ds$. Based on this, one can associate to the kinematic model (2) its geometric counterpart

$$\boldsymbol{q}' = \tilde{\boldsymbol{v}}, \tag{5}$$

where $\tilde{\boldsymbol{v}}$ is a vector of geometric inputs that are related to the actual inputs $\boldsymbol{v}$ by the relationship

$$\boldsymbol{v} = \dot{s}\, \tilde{\boldsymbol{v}}.$$

[1] It is assumed that holonomic constraints, if present, have been either solved (through appropriate choice of the generalized coordinates) or incorporated in the task definition.
[2] Avoidance of self-collisions and joint limits can be easily incorporated in this constraint and in fact they are included in our implementation (see Section 7).

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

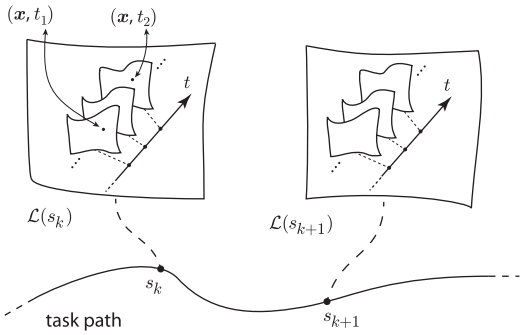


Fig. 1. The structure of $\mathcal{S}_{\text{task}}$: each leaf $\mathcal{L}(s)$ is the set of states $\boldsymbol{x}$ satisfying the task constraint at s, replicated along the time axis. For the kinematic case, we have $\boldsymbol{x} = \boldsymbol{q}$, while for the dynamic case $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$.

3.2. Search space

For a kinematic system, the state space coincides with the configuration space $\mathcal{C}$. In the presence of moving obstacles, however, the planning space for K_TCMP_MO cannot be a simple subset of $\mathcal{C}$. A configuration may be, in fact, admissible at a certain time instant and not admissible at another due to the movement of the obstacles. Hence, time must be explicitly introduced in the picture.

Define

- the *configuration-time space* (henceforth CTS) as $\mathcal{S} = \mathcal{C} \times [0, \infty)$;
- the *occupied CTS* as $\mathcal{S}_{\text{occ}} = \{(\boldsymbol{q}, t) \in \mathcal{S} : \mathcal{R}(\boldsymbol{q}(t)) \cap \mathcal{O}(t) \neq \emptyset\}$;
- the *free CTS* as $\mathcal{S}_{\text{free}} = \mathcal{S} \setminus \mathcal{S}_{\text{occ}}$;
- the *task-constrained CTS* as $\mathcal{S}_{\text{task}} = \{(\boldsymbol{q}, t) \in \mathcal{S} : \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{y}_d(s), \text{ for some } s \in [s_{\text{ini}}, s_{\text{fin}}]\}$.

In $\mathcal{S}_{\text{task}}$, t is irrelevant; i.e., for any point in $\mathcal{S}_{\text{task}}$, there exist infinite other points with the same $\boldsymbol{q}$ and different time instant $t \in [0, \infty)$.

The set $\mathcal{S}_{\text{task}}$ is a manifold with boundary that naturally decomposes as a foliation:

$$\mathcal{S}_{\text{task}} = \bigcup_{s \in [s_{\text{ini}}, s_{\text{fin}}]} \mathcal{L}(s), \tag{6}$$

with the generic *leaf* defined as

$$\mathcal{L}(s) = \{(\boldsymbol{q}, t) \in \mathcal{S} : \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{y}_d(s)\}.$$

On each leaf, t can assume any value in $[0, \infty)$. Figure 1 illustrates the structure of $\mathcal{S}_{\text{task}}$: the set of states (in this case, configurations) satisfying the task constraint at s is replicated along the time axis to form a leaf of $\mathcal{S}_{\text{task}}$.

The existence of a solution to the K_TCMP_MO problem depends on the motion of the obstacles, and in particular, on the connectedness of $\mathcal{S}_{\text{task}} \cap \mathcal{S}_{\text{free}}$, i.e., the portion of the free CTS that is compatible with the task path constraint. However, candidate solutions contained in such space must still satisfy the bounds on generalized velocities to be feasible.

3.3. K_TCMP_MO planner

Our planner builds a tree in the search space $\mathcal{S}_{\text{task}} \cap \mathcal{S}_{\text{free}}$. In doing so, it makes use of N samples of the assigned task path, denoted by $\boldsymbol{y}_k = \boldsymbol{y}_d(s_k)$, corresponding to a predefined sequence $\{s_1 = s_{\text{ini}}, \ldots, s_k, \ldots, s_N = s_{\text{fin}}\}$ of values of s. Denote by $\mathcal{L}_k = \mathcal{L}(s_k)$ be the leaf associated to $\boldsymbol{y}_k$ (see Fig. 1).

The root of the tree is the pair $(\boldsymbol{q}_{\text{ini}}, 0)$, consisting of the initial robot state and time instant. This will be the only vertex on $\mathcal{L}_1$. All the other vertexes lie on some $\mathcal{L}_k$, $k = 2, \ldots, N$; in principle, there will be several vertexes on each leaf. Each vertex is a pair $(\boldsymbol{q}, t)$ representing a robot state and the time at which it was attained. An edge is a feasible collision-free subtrajectory between two vertexes lying on adjacent leaves.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

3.3.1. Motion generation. Starting from a generic vertex of the tree located on a certain leaf, the motion generation scheme attempts to build a feasible subtrajectory that is contained in $\mathcal{S}_{\text{task}} \cap \mathcal{S}_{\text{free}}$ and lands on an adjacent leaf, which may be the next or the previous. The proposed scheme extends that of ref. [21] so as to plan a time history $s(t)$ in addition to a geometric motion $\boldsymbol{q}(s)$. In particular, we separately generate $\dot{s}$ and $\boldsymbol{q}'$, and we rely on (4) to guarantee that the resulting subtrajectory is feasible.

Consider vertex $V = (\boldsymbol{q}_V, t_V)$ located on leaf $\mathcal{L}_k$, where $s = s_k$. For the time history, a constant $\dot{s}$ is chosen for the whole subtrajectory:

$$\dot{s} \in [-b_{\max}, b_{\max}], \tag{7}$$

where $b_{\max} > 0$. On the subtrajectory, the motion of s over t will then be linear and go from s_k to s_{k+1} if $\dot{s} > 0$ (*forward motion*), from s_k to s_{k-1} if $\dot{s} < 0$ (*backward motion*). Once subtrajectories are patched together in a solution, we will obtain a piecewise-constant profile for $\dot{s}$, and a continuous, piecewise-linear time history for the path parameter $s(t)$.

Let us now discuss the generation of $\boldsymbol{q}'$ on the subtrajectory. Differentiating (1) with respect to s and using (5) gives

$$\boldsymbol{y}' = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{q}' = \boldsymbol{J}(\boldsymbol{q})\tilde{\boldsymbol{v}},$$

where $\boldsymbol{J} = \partial \boldsymbol{f}/\partial \boldsymbol{q}$ is the $m \times n$ task Jacobian. Exact task execution over the whole subtrajectory is then guaranteed by letting

$$\tilde{\boldsymbol{v}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})(\pm \boldsymbol{y}'_d + k_p \boldsymbol{e}_y) + (\boldsymbol{I}_n - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}))\tilde{\boldsymbol{w}}, \tag{8}$$

where $\boldsymbol{J}^{\dagger} = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}$ is the pseudoinverse matrix of $\boldsymbol{J}$, k_p is a positive gain, $\boldsymbol{e}_y = \boldsymbol{y}_d - \boldsymbol{y}$ is the task error, $\boldsymbol{I}_n - \boldsymbol{J}^{\dagger}\boldsymbol{J}$ is the orthogonal projection matrix in the null space of $\boldsymbol{J}$, and $\tilde{\boldsymbol{w}}$ is an arbitrary n-dimensional *residual input* that produces internal motions without affecting the task. Note that the sign in front of $\boldsymbol{y}'_d$ should be taken positive for a forward motion and negative for a backward motion.

The subpath $\boldsymbol{q}(s)$ is then generated by plugging (8) in (5) and integrating numerically from s_k to s_{k+1} for a forward motion, and from s_k to s_{k-1} for a backward motion.

By adding the time history to the subpath, the resulting subtrajectory is reconstructed. In doing so, both its feasibility and the avoidance of obstacles are continuously checked. If one of these two conditions is violated, motion generation is prematurely terminated. Otherwise, integration proceeds until the subtrajectory lands on an adjacent leaf to $\mathcal{L}_k$, either $\mathcal{L}_{k+1}$ or $\mathcal{L}_{k-1}$.

Figure 2 illustrates how motion generation works. Once a vertex V on $\mathcal{L}_k$ has been selected for extension, different choices of $\dot{s} > 0$ in (7) for the same $\boldsymbol{q}'$ would produce different motions on the same geometric path. Both motions terminate in the same configuration on $\mathcal{L}_{k+1}$, but at different time instants.

3.3.2. Tree extension. Tree extension uses an RRT-like mechanism. At each iteration, a random task sample $\boldsymbol{y}_{\text{rand}}$ is extracted from the predefined sequence $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$ and an associated inverse kinematic solution $\boldsymbol{q}_{\text{rand}} = \boldsymbol{f}^{-1}(\boldsymbol{y}_{\text{rand}})$ is computed; this is done by randomly choosing the values of $n - m$ generalized coordinates at $\boldsymbol{q}_{\text{rand}}$ and then solving for the remaining coordinates. A time instant t_{rand} is then attached to $\boldsymbol{q}_{\text{rand}}$ by random choice in $[0, t_{\max}]$, where $t_{\max}$ denotes the largest time associated to a vertex in the current tree. By construction, $(\boldsymbol{q}_{\text{rand}}, t_{\text{rand}})$ is a sample of $\mathcal{S}_{\text{task}}$.

The tree is then searched to find the vertex $V = (\boldsymbol{q}_{\text{near}}, t_{\text{near}})$ that is closest to $(\boldsymbol{q}_{\text{rand}}, t_{\text{rand}})$, using a weighted metric that accounts for distances in the CTS $\mathcal{S}$. Call s_k and $\mathcal{L}_k$ the parameter value and leaf, respectively, corresponding to $\boldsymbol{q}_{\text{near}}$. At this point, the tree is extended from V via the following procedure:

1. Randomly choose two values of $\dot{s}$ as in (7), one positive (forward motion) and one negative (backward motion). Compute the corresponding time histories, stopping at $s = s_{k+1}$ for the forward motion and at $s = s_{k-1}$ for the backward motion.
2. Randomly choose r values of $\tilde{\boldsymbol{w}}$ in (8) inside a bounded subset W of $\boldsymbol{R}^n$.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

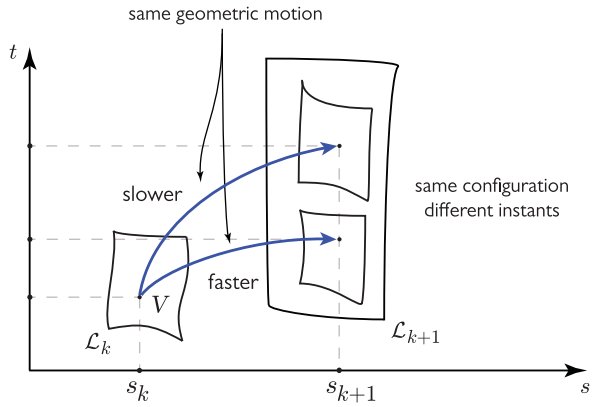


Fig. 2. Generation of forward motions in K_TCMP_MO: different choices of $\dot{s} > 0$ for the same $\boldsymbol{q}'$ produce the same geometric motion at different speeds. The same is true for backward motions associated to $\dot{s} < 0$ (not shown).

3. For each value of $\tilde{\boldsymbol{w}}$, generate a forward subpath from $\mathcal{L}_k$ to $\mathcal{L}_{k+1}$ and a backward subpath from $\mathcal{L}_k$ to $\mathcal{L}_{k-1}$ by integrating (5)–(8) from s_k to s_{k+1} and from s_k to s_{k-1}, respectively. Discard the subpath if a configuration is met where $\boldsymbol{J}$ loses rank (or is close to losing rank).
4. For each category (forward/backward), retain only the subpath terminating closest to $\boldsymbol{q}_{\text{rand}}$.
5. For each subpath, add the appropriate (forward/backward) time history in order to reconstruct the resulting subtrajectory, and check it for feasibility in terms of generalized velocities (using (4)) as well as avoidance of obstacles. Discard the subpath as soon as one of these conditions is violated.

At the end of the procedure, a pair (edge,vertex) originating from V is added to the tree for each subtrajectory (two at most) that is feasible and does not cause collisions.

We conclude this subsection with a couple of remarks.

- As an interesting alternative to the purely randomized generation of $\tilde{\boldsymbol{w}}$ in Step 2, one may take an exploration–exploitation perspective. This approach is discussed in detail in Section 6.
- While exact task execution is ensured by the use of (8), generalized velocity bounds are taken into account in Step 5 of the previous procedure by *validation* of the generated subtrajectories, i.e., by checking whether a candidate subpath satisfies the bound and rejecting it otherwise. A different option is to guarantee satisfaction of these bounds by *construction*; to this end, one can modify Step 1 by (a) postponing it after Step 4, and (b) choosing $b_{\max}$ in (7) in such a way that the bounds are certainly respected. In view of (4), this can be achieved by letting

$$b_{\max} = \min_{i=1,\dots,n} \frac{v_i^M}{q'_{i,\max}}, \tag{9}$$

where[3] $q'_{i,\max} = \max_{s\in\mathcal{I}} |q'_i(s)|$, with $\mathcal{I} = [s_k, s_{k+1}]$ for a forward subpath and $\mathcal{I} = [s_k, s_{k-1}]$ for a backward subpath. This modification obviously increases the computational load of the planner, but can also be expected to improve its efficiency thanks to a reduced number of rejected motions.

See the Appendix for a pseudocode description of the K_TCMP_MO planner.

3.4. Extension: Non-holonomic systems

As an extension of the kinematic case, we now remove the free-flying assumption and consider a system that is subject to a set of h Pfaffian non-holonomic constraints. In this case, the number of degrees of freedom (i.e., independent infinitesimal motions) decreases to $p = n - h$.

[3]Note that executing Step 1 after Step 4 guarantees that the geometric subpaths have been already generated and therefore these maximum values can be computed.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

In a general robotic system, the non-holonomic constraints may involve all configuration variables or only a subset of them. To take this into account, reorder and partition the configuration vector $\boldsymbol{q}$ as $(\boldsymbol{q}_c, \boldsymbol{q}_u)$, where the subvector of *constrained* coordinates $\boldsymbol{q}_c$ is n_c-dimensional and the subvector of *unconstrained* coordinates $\boldsymbol{q}_u$ is n_u-dimensional. The non-holonomic constraints only act on the $\boldsymbol{q}_c$ subvector:

$$\boldsymbol{A}(\boldsymbol{q}_c)\dot{\boldsymbol{q}}_c = \boldsymbol{0},$$

with the $h \times n_c$ constraint matrix $\boldsymbol{A}$ assumed to be always full row rank. Note that we have $0 < n_c \leq n$ and correspondingly $0 \leq n_u < n$.

In the light of the above, the motion model used for planning can be expressed as

$$\dot{\boldsymbol{q}} = \begin{pmatrix} \dot{\boldsymbol{q}}_c \\ \dot{\boldsymbol{q}}_u \end{pmatrix} = \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}_c)\boldsymbol{v}_c \\ \boldsymbol{v}_u \end{pmatrix} = \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}_c) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{n_u} \end{pmatrix} \boldsymbol{v}, \tag{10}$$

where the columns of the $n_c \times (n_c - h)$ matrix $\boldsymbol{G}(\boldsymbol{q}_c)$ are a basis for the null space of $\boldsymbol{A}(\boldsymbol{q}_c)$. The velocity input vectors $\boldsymbol{v}_c$ and $\boldsymbol{v}_u$ are $(n_c - h)$-dimensional and n_u-dimensional, respectively, and they are collected in the p-dimensional vector $\boldsymbol{v} = (\boldsymbol{v}_c, \boldsymbol{v}_u)$. Note that the components of $\boldsymbol{v}_c$ are actually *pseudovelocities*, whereas $\boldsymbol{v}_u$ contains true generalized velocities. All are subject to bounds:

$$|v_{c,i}| \leq v_{c,i}^M, \quad i = 1, \ldots, n_c - h,$$
$$|v_{u,j}| \leq v_{u,j}^M, \quad j = 1, \ldots, n_u,$$

which can be put in a compact form identical to (3):

$$|\boldsymbol{v}| \leq \boldsymbol{v}^M.$$

The kinematic model (10) applies to most mobile robots, including single-body non-holonomic mobile robots ($n_u = 0$) and non-holonomic mobile manipulators ($n_u > 0$).

In the nonholonomic context, the formulation of the K_TCMP_MO problem is the same of Section 3.1, with only two differences. Condition 1 must be expressed directly on the velocity input vectors:

C1, non-holonomic case: $|\boldsymbol{v}| \leq \boldsymbol{v}^M$, for $t \in [0, T]$.

Moreover, Assumption 1 must be slightly rephrased:

A1, non-holonomic case: The robot is kinematically redundant for the task $\boldsymbol{y}$, i.e., $p > m$ (the number of degrees of freedom exceeds the task dimension).

The structure of the search space is the same discussed in Section 3.2, because non-holonomic constraints affect local mobility but preserve global accessibility.

Inside the planner, motion generation needs an adaptation. Recalling the separation of generalized velocities entailed by (4), the geometric counterpart of (10) can be written as

$$\boldsymbol{q}' = \begin{pmatrix} \boldsymbol{q}'_c \\ \boldsymbol{q}'_u \end{pmatrix} = \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}_c)\,\tilde{\boldsymbol{v}}_c \\ \tilde{\boldsymbol{v}}_u \end{pmatrix} = \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}_c) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{n_u} \end{pmatrix} \tilde{\boldsymbol{v}}, \tag{11}$$

Using (11), we obtain

$$\boldsymbol{y}' = \boldsymbol{J}(\boldsymbol{q})\,\boldsymbol{q}' = \begin{pmatrix} \boldsymbol{J}_c(\boldsymbol{q}) & \boldsymbol{J}_u(\boldsymbol{q}) \end{pmatrix} \begin{pmatrix} \boldsymbol{q}'_c \\ \boldsymbol{q}'_u \end{pmatrix} = \begin{pmatrix} \boldsymbol{J}_c(\boldsymbol{q})\,\boldsymbol{G}(\boldsymbol{q}_c) & \boldsymbol{J}_u(\boldsymbol{q}) \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{v}}_c \\ \tilde{\boldsymbol{v}}_u \end{pmatrix} = \boldsymbol{J}_{\mathrm{nh}}(\boldsymbol{q})\,\tilde{\boldsymbol{v}}, \tag{12}$$

with the actual inputs still expressed as $\boldsymbol{v} = \dot{s}\,\tilde{\boldsymbol{v}}$. The $m \times p$ matrix $\boldsymbol{J}_{\mathrm{nh}}(\boldsymbol{q})$ is called the *non-holonomic task Jacobian* in the following.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

The motion generation formula for the non-holonomic case is then simply obtained by replacing the standard task Jacobian $\boldsymbol{J}$ with its non-holonomic version $\boldsymbol{J}_{\rm nh}$ in (8):

$$\tilde{\boldsymbol{v}} = \begin{pmatrix} \tilde{\boldsymbol{v}}_c \\ \tilde{\boldsymbol{v}}_u \end{pmatrix} = \boldsymbol{J}_{\rm nh}^{\dagger}(\boldsymbol{q})(\pm \boldsymbol{y}_d' + k_p \boldsymbol{e}_y) + (\boldsymbol{I}_p - \boldsymbol{J}_{\rm nh}^{\dagger}(\boldsymbol{q})\boldsymbol{J}_{\rm nh}(\boldsymbol{q}))\, \tilde{\boldsymbol{w}}_{\rm nh}, \tag{13}$$

where $\tilde{\boldsymbol{w}}_{\rm nh}$ is now an arbitrary p-vector.

With this preamble, the motion planner is exactly the same of the previous section, with the only adjustment that now $\tilde{\boldsymbol{w}}_{\rm nh}$ in Step 2 of the extension procedure is randomly chosen in a bounded subset W of $\boldsymbol{R}^p$.

Finally, also in this case, it is possible to guarantee that the generated subtrajectories are feasible by construction (rather than by validation) by postponing Step 1 after Step 4 and choosing the bound in (7) as

$$b_{\max} = \min_{\substack{i=1,\ldots,n_c \\ j=1,\ldots,n_u}} \left\{ \frac{v_{c,i}^M}{\tilde{v}_{i,\max}}, \frac{v_{u,j}^M}{q'_{j,\max}} \right\}, \tag{14}$$

where $\tilde{v}_{i,\max} = \max_{s\in\mathcal{I}} |\tilde{v}_i(s)|$, $q'_{j,\max} = \max_{s\in\mathcal{I}} |q'_j(s)|$, and $\mathcal{I} = [s_k, s_{k+1}]$ for a forward subpath, $\mathcal{I} = [s_k, s_{k-1}]$ for a backward subpath.

4. The Dynamic Case

In the second version of the problem, we consider a fully actuated, dynamic-level model of the robot. We assume that the robot is free of constraints and that the generalized force inputs are bounded. This is an adequate representation for robots with articulated limbs, such as manipulators, or for dynamically consistent digital animation.

4.1. Problem formulation

The robot dynamics are expressed in the Euler–Lagrange form:

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau},$$

where $\boldsymbol{B}(\boldsymbol{q})$ is the inertia matrix, $\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ collects velocity and gravitational terms, and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$ are the generalized forces provided by the actuators. This leads to the dynamic-level model

$$\ddot{\boldsymbol{q}} = \boldsymbol{a}, \tag{15}$$

where $\boldsymbol{a} = (a_1, \ldots, a_n)$ represents the vector of generalized acceleration inputs, with bounds on the generalized forces

$$|\boldsymbol{B}(\boldsymbol{q})\boldsymbol{a} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})| \le \boldsymbol{\tau}^M, \tag{16}$$

as well as on the generalized velocities

$$|\dot{\boldsymbol{q}}| \le \dot{\boldsymbol{q}}^M.$$

These limit, respectively, the robot acceleration/deceleration along the path and its velocity (forward or backward). To be feasible, a trajectory in $\mathcal{C}$ must now satisfy both types of bounds.

Under assumptions A1–A3, a solution to the *dynamic version* of the TCMP_MO problem (called D_TCMP_MO in the following) consists of a path $\boldsymbol{q}(s) : [s_{\rm ini}, s_{\rm fin}] \mapsto \mathcal{C}$ and a continuous time history $s(t) : [0, T] \mapsto [s_{\rm ini}, s_{\rm fin}]$ such that

C1: $|\dot{\boldsymbol{q}}(t)| \le \dot{\boldsymbol{q}}^M$ and $|\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})| \le \boldsymbol{\tau}^M$, for $t \in [0, T]$;
C2: $s(0) = s_{\rm ini}$ and $s(T) = s_{\rm fin}$;
C3: $\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{q}(s(t))) = \boldsymbol{y}_d(s(t))$ for all $t \in [0, T]$;

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

C4: $\boldsymbol{q}(s_{\text{ini}}) = \boldsymbol{q}_{\text{ini}}, \dot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}_{\text{ini}}$;
C5: $\mathcal{R}(\boldsymbol{q}(s(t))) \cap \mathcal{O}(t) = \emptyset$, for all $t \in [0, T]$.

In comparison with the formulation of the K_TCMP_MO problem in Section 3.1, C1 has been adapted to the new definition of feasibility, while C4 accounts for the fact that the new robot state is $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and therefore the initial generalized velocity (typically zero) is also assigned. Conditions 2, 3, and 5 are unchanged.

The separation of $\dot{\boldsymbol{q}}$ entailed by Eq. (4) still holds and induces the following form for the generalized accelerations involved in C1:

$$\ddot{\boldsymbol{q}} = \dot{s}^2 \boldsymbol{q}'' + \ddot{s} \boldsymbol{q}'. \tag{17}$$

One can associate then to the dynamic model (15) its geometric counterpart

$$\boldsymbol{q}'' = \tilde{\boldsymbol{a}}, \tag{18}$$

where $\tilde{\boldsymbol{a}}$ is a vector of geometric inputs that are related to the actual inputs $\boldsymbol{a}$ by the relationship

$$\boldsymbol{a} = \dot{s}^2 \tilde{\boldsymbol{a}} + \ddot{s} \boldsymbol{q}'.$$

4.2. Search space

When discussing the search space for the D_TCMP_MO problem, the main difference with respect to K_TCMP_MO is the fact that the robot state space is now $\mathcal{X} = \mathcal{C} \times T_{\boldsymbol{q}} \mathcal{C}$, where $T_{\boldsymbol{q}} \mathcal{C}$ denotes the tangent space of $\mathcal{C}$ at $\boldsymbol{q}$. This leads us directly to the following definitions (compare with Section 4.2):

- The *state-time space* (henceforth STS) as $\mathcal{S} = \mathcal{X} \times [0, \infty)$.
- The *occupied STS* as $\mathcal{S}_{\text{occ}} = \{(\boldsymbol{q}, t) \in \mathcal{S} : \mathcal{R}(\boldsymbol{q}(t)) \cap \mathcal{O}(t) \neq \emptyset\}$.
- The *free STS* as $\mathcal{S}_{\text{free}} = \mathcal{S} \setminus \mathcal{S}_{\text{occ}}$.
- The *task-constrained STS* as $\mathcal{S}_{\text{task}} = \{(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \in \mathcal{S} : \boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{y}_d(s), \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \dot{s}\boldsymbol{y}_d'(s)$, for some $s \in [s_{\text{ini}}, s_{\text{fin}}], \dot{s} \in (-\infty, \infty)\}$.

$\mathcal{S}_{\text{task}}$ still has the structure of a foliation as expressed by (6) and shown by Fig. 1, with the leaves now defined as

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

As before, the existence of a solution to the planning problem depends on the interplay between the task path and the obstacles' motion, i.e., on the connectedness of the search space $\mathcal{S}_{\text{task}} \cap \mathcal{S}_{\text{free}}$. A candidate trajectory contained in such space may still turn out to be unfeasible, because of the bounds on generalized velocities and (now also) forces.

4.3. D_TCMP_MO planner

Similarly to K_TCMP_MO, the D_TCMP_MO planner builds a tree in $\mathcal{S}_{\text{task}} \cap \mathcal{S}_{\text{free}}$ and uses a sequence of samples of the assigned task path, denoted by $\boldsymbol{y}_k = \boldsymbol{y}_d(s_k)$, $k = 1, \ldots, N$, each with its corresponding leaf $\mathcal{L}_k$ of $\mathcal{S}_{\text{task}}$ (see Fig. 1).

The root of the tree, and the only vertex on $\mathcal{L}_1$, is the triple $(\boldsymbol{q}_{\text{ini}}, \dot{\boldsymbol{q}}_{\text{ini}}, 0)$, consisting of the initial robot state and time instant. Any other vertex lies on some $\mathcal{L}_k$, $k = 2, \ldots, N$, and is a triple $(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)$ representing a robot state and the time at which it was attained.

4.3.1. Motion generation. Recall that the objective of motion generation is to start from a vertex of the tree located on a certain leaf and to produce a feasible subtrajectory which is contained in $\mathcal{S}_{\text{task}} \cap \mathcal{S}_{\text{free}}$ and lands on either the next or the previous leaf. As before, we plan a geometric motion $\boldsymbol{q}(s)$ as well as a time history $s(t)$. In particular, $\ddot{s}$ and $\boldsymbol{q}''$ are generated separately, and feasibility of the resulting trajectory is guaranteed by appropriately using (17).

Consider vertex $V = (\boldsymbol{q}_V, \dot{\boldsymbol{q}}_V, t_V)$ located on leaf $\mathcal{L}_k$, where $s = s_k$. For the time history, a constant $\ddot{s}$ is chosen for the whole subtrajectory:

$$\ddot{s} \in [-c_{\max}, c_{\max}], \tag{19}$$

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

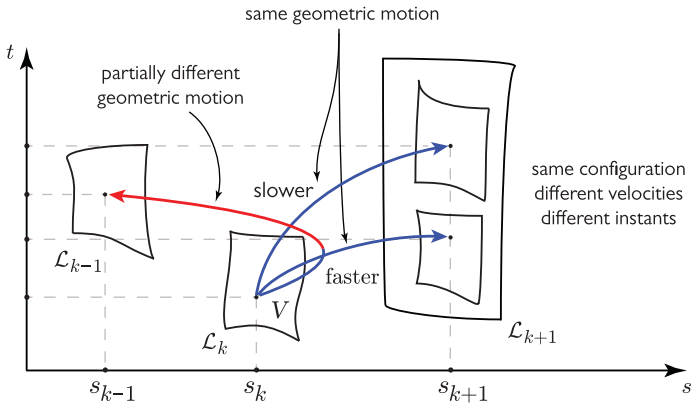


Fig. 3. An illustration of motion generation within D_TCMP_MO. In this particular case, it is assumed that $\dot{s}$ at V is positive. The two forward motions (blue) correspond to the same choice of $\tilde{\boldsymbol{z}}$ but different positive values of $\ddot{s}$. The decelerating motion (red) is generated by the same $\tilde{\boldsymbol{z}}$ but now with a negative $\ddot{s}$. The geometric motion is the same of the forward case until the direction is reversed. Compare with Fig. 2.

where $c_{\max} > 0$. A positive $\ddot{s}$ will give an *accelerating motion*, whereas a negative $\ddot{s}$ will give an *decelerating motion*. The profile of s in t will then be quadratic over the subtrajectory. In particular, depending on the sign of the initial value of $\dot{s}$ and the chosen $\ddot{s}$, we will obtain four kinds of motions of s over t: (1) a forward motion from s_k to s_{k+1}, either accelerating or decelerating, (2) a decelerating motion where s proceeds initially forward from s_k but then reverses its direction and proceeds backwards until s_{k-1}, (3) a backward motion from s_k to s_{k-1}, either accelerating or decelerating, (4) an accelerating motion where s proceeds initially backward from s_k but then reverses its direction and proceeds forward until s_{k+1}. When subtrajectories are patched together in a solution, we will obtain a piecewise-constant profile for $\ddot{s}$, a continuous piecewise-linear profile for $\dot{s}$ and correspondingly a continuous, piecewise-quadratic time history for the path parameter $s(t)$.

As for the generation of $\boldsymbol{q}'$, we use (18) to write

$$\boldsymbol{y}'' = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{q}'' + \boldsymbol{J}'(\boldsymbol{q}, \boldsymbol{q}')\boldsymbol{q}' = \boldsymbol{J}(\boldsymbol{q})\tilde{\boldsymbol{a}} + \boldsymbol{J}'(\boldsymbol{q}, \boldsymbol{q}')\boldsymbol{q}',$$

and thus we can guarantee task execution by letting

$$\tilde{\boldsymbol{a}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})(\boldsymbol{y}_d'' - \boldsymbol{J}'(\boldsymbol{q}, \boldsymbol{q}')\,\boldsymbol{q}' + k_p\,\boldsymbol{e}_y + k_d\,\boldsymbol{e}_y') + (\boldsymbol{I}_n - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}))\tilde{\boldsymbol{z}}, \tag{20}$$

where k_d is another positive gain and $\tilde{\boldsymbol{z}}$ is an n-dimensional residual input vector which can be chosen arbitrarily without effect on the task; all the other quantities have already been defined with reference to (8). Note that $\boldsymbol{e}_y' = \pm\boldsymbol{y}_d' - \boldsymbol{J}\boldsymbol{q}'$, where the positive (negative) sign must be used in correspondence of increasing (decreasing) values of s, i.e., during a forward (backward) motion phase.

The geometric path is then generated by plugging Eq. (20) in (18) and integrating numerically over s from s_k according to the specific profile of $s(t)$ associated to the chosen $\ddot{s}$.

By combining the time history with the geometric subpath, the resulting subtrajectory is reconstructed. In doing so, we continuously check its feasibility (in terms of the bounds on the generalized velocities and forces) as well as the avoidance of moving obstacles. If either of these is violated, generation of the current subtrajectory is aborted. Otherwise, integration stops when the subtrajectory lands on an adjacent leaf to $\mathcal{L}_k$.

Figure 3 illustrates some typical situations encountered when applying motion generation from a vertex on $\mathcal{L}_k$.

4.3.2. Tree extension. Extension of the planning tree occurs similarly to the kinematic case. At each iteration, a random task sample $\boldsymbol{y}_{\text{rand}}$ is extracted from the predefined sequence $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$ and an inverse solution $\boldsymbol{q}_{\text{rand}}$ is computed. A random task-consistent generalized velocity $\dot{\boldsymbol{q}}_{\text{rand}}$ is then chosen and paired to $\boldsymbol{q}_{\text{rand}}$. Finally, a time instant t_{rand} is sampled from $[0, t_{\max}]$ and attached to the random state. By construction, $(\boldsymbol{q}_{\text{rand}}, \dot{\boldsymbol{q}}_{\text{rand}}, t_{\text{rand}})$ is a sample of $\mathcal{S}_{\text{task}}$.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

At this point, the tree is searched for the closest vertex to $(\boldsymbol{q}_{\text{rand}}, \dot{\boldsymbol{q}}_{\text{rand}}, t_{\text{rand}})$, using a weighted metric accounting for distances in the STS $\mathcal{S}$. Denote this vertex by $V = (\boldsymbol{q}_{\text{near}}, \dot{\boldsymbol{q}}_{\text{near}}, t_{\text{near}})$, and say it is located on leaf $\mathcal{L}_k$. The tree is then extended from V via the following procedure:

1. Randomly choose two values of $\ddot{s}$ from Eq. (19), one positive (accelerating motion) and one negative (decelerating motion). Compute the corresponding time histories $s(t)$, stopping as soon as either $\mathcal{L}_{k+1}$ or $\mathcal{L}_{k-1}$ is reached.
2. Randomly choose r values of $\tilde{\boldsymbol{z}}$ in Eq. (20) inside a bounded subset Z of $\boldsymbol{R}^n$.
3. For each value of $\tilde{\boldsymbol{z}}$, generate an accelerating subpath and a decelerating subpath by integrating Eqs. (18)–(20) along the profile of s corresponding to the two time histories. Discard the subpath if a configuration is met where $\boldsymbol{J}$ loses rank (or is close to losing rank).
4. For each category (accelerating/decelerating), retain only the subpath terminating closest to $\boldsymbol{q}_{\text{rand}}$.
5. For each subpath, add the appropriate (accelerating/decelerating) time history in order to reconstruct the resulting subtrajectory, and check it for feasibility in terms of generalized velocities and forces (using (16) and (17)) as well as for avoidance of the moving obstacles. Discard the subpath as soon as one of these conditions is violated.

At the end of the procedure, a pair (edge, vertex) originating from V is added to the tree for each subtrajectory (two at most) that is feasible and does not cause collisions. In conclusion, note the following facts:

- Also in this case, one may take an exploration–exploitation approach as an alternative to the purely randomized generation of $\tilde{\boldsymbol{z}}$ of Step 1, see Section 6.
- In the dynamic case, modifying the extension procedure so as to enforce feasibility of the generated subtrajectories (without the need of checking them) is more difficult than in the kinematic case, essentially because the complete separation at the velocity level between geometric path and time history entailed by (4) does not carry on to the acceleration level, as shown by (17). However, a similar result can be achieved by building the subtrajectory via numerical integration in t rather than in s, and choosing a different value of $\ddot{s}$ at each integration instant, rather than a constant $\ddot{s}$ for the whole subtrajectory. In particular, use (17) to rewrite (16) as

$$|\boldsymbol{B}(\boldsymbol{q})(\dot{s}^2\,\boldsymbol{q}'' + \ddot{s}\,\boldsymbol{q}') + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})| \leq \boldsymbol{\tau}^M. \tag{21}$$

  Being

$$|\boldsymbol{B}(\boldsymbol{q})(\dot{s}^2\,\boldsymbol{q}'' + \ddot{s}\,\boldsymbol{q}') + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})| \leq |\ddot{s}||\boldsymbol{B}(\boldsymbol{q})\boldsymbol{q}'| + \dot{s}^2|\boldsymbol{B}(\boldsymbol{q})\boldsymbol{q}''| + |\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})|,$$

  we can guarantee that the generalized force bounds (21) are verified at the current instant by letting[4]

$$c_{\max} = \min_{i=1,\ldots,n} \frac{\tau_i^M - |\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}})|_i - \dot{s}^2|\boldsymbol{B}(\boldsymbol{q})\boldsymbol{q}''|_i}{|\boldsymbol{B}(\boldsymbol{q})\boldsymbol{q}'|_i} \tag{22}$$

  and using this value in (19) to generate a new $\ddot{s}$ at each integration instant. Considering that numerical integration is typically performed with very small stepsizes, one can expect the generalized forces to be within their bounds at all times.

5. Probabilistic Completeness

We now prove that the proposed planners are probabilistically complete, showing that the random choice of the residual input vector ($\tilde{\boldsymbol{w}}$ or $\tilde{\boldsymbol{a}}$) is instrumental to this purpose.

Proposition 1. *Consider a TCMP_MO problem, either in the kinematic or in the dynamic version, and assume that a solution trajectory $\boldsymbol{q}^*(t)$ exists which can be produced by the appropriate planner (K_TCMP_MO or D_TCMP_MO). Then, the probability of generating such path converges to one as the number of iteration increases.*

[4]Here, $|\cdot|_i$ indicates the absolute value of the ith component of a vector.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

Proof. First of all, note that the solution trajectory $\boldsymbol{q}^*(t)$ actually consists of two components: a path $\boldsymbol{q}^*(s) : [s_{\text{ini}}, s_{\text{fin}}] \mapsto \mathcal{C}$ and a time history $s^*(t) : [0, T] \mapsto [s_{\text{ini}}, s_{\text{fin}}]$. We must then prove that the planner is able to generate both. Let us consider the kinematic case first.

For compactness, denote by $\boldsymbol{q}^*_{\rightarrow}$ the solution path $\boldsymbol{q}^*(s) : [s_{\text{ini}}, s_{\text{fin}}]$. By assumption, $\boldsymbol{q}^*_{\rightarrow}$ is a path in $\mathcal{S}_{\text{task}} \cap \mathcal{S}_{\text{free}}$ obtained by integrating (8) from $s_1 = s_{\text{ini}}$ to $s_N = s_{\text{fin}}$ under a piecewise-constant sequence of residual input vectors

$$\tilde{\boldsymbol{w}}^*_{\rightarrow} = \{\tilde{\boldsymbol{w}}^*_1, \ldots, \tilde{\boldsymbol{w}}^*_{N-1}\},$$

with $\tilde{\boldsymbol{w}} = \tilde{\boldsymbol{w}}^*_k$ for $s \in [s_k, s_{k+1})$. Based on the properties of solutions of ordinary differential equations, $\boldsymbol{q}^*_{\rightarrow}$ is a continuous function of $\boldsymbol{q}_{\text{ini}}$ (the initial condition, which is given) as well as of $\tilde{\boldsymbol{w}}^*_{\rightarrow}$ (the residual input sequence, which is generated by the planner).

Define the *collision clearance* of $\boldsymbol{q}^*_{\rightarrow}$ as

$$\gamma = \min_{t \in [0,T]} d\,(\boldsymbol{q}^*(s^*(t)), t),$$

where $d\,(\boldsymbol{q}, t)$ denotes the minimum distance between configuration $\boldsymbol{q}$ and the $\mathcal{C}$-obstacle region at time t. At this point, we can define a tube in $\mathcal{C}$ around $\boldsymbol{q}^*_{\rightarrow}$, denoted by $\mathcal{T}_{\mathcal{C}}(\boldsymbol{q}^*_{\rightarrow})$, whose cross-section has radius γ. By definition, any path that goes from $\mathcal{L}_1$ to $\mathcal{L}_N$ and is contained in $\mathcal{T}_{\mathcal{C}}(\boldsymbol{q}^*_{\rightarrow})$ realizes the whole task and is collision-free (i.e., satisfies conditions C2–C5 of the K_TCMP_MO problem).

We can now rely on the continuity of $\boldsymbol{q}^*_{\rightarrow}$ with respect to $\tilde{\boldsymbol{w}}^*_1, \ldots, \tilde{\boldsymbol{w}}^*_{N-1}$ to claim that there exists a collection of neighborhoods in $\boldsymbol{R}^n$

$$S(\tilde{\boldsymbol{w}}^*_{\rightarrow}) = \{S(\tilde{\boldsymbol{w}}^*_1), \ldots, S(\tilde{\boldsymbol{w}}^*_{N-1})\},$$

such that any choice of $\tilde{\boldsymbol{w}}_{\rightarrow}$ in $S(\tilde{\boldsymbol{w}}^*_{\rightarrow})$ (i.e., of $\tilde{\boldsymbol{w}}_1$ in $S(\tilde{\boldsymbol{w}}^*_1)$, of $\tilde{\boldsymbol{w}}_2$ in $S(\tilde{\boldsymbol{w}}^*_2)$, and so on) produces a path $\boldsymbol{q}_{\rightarrow}$ that goes from $\mathcal{L}_1$ to $\mathcal{L}_N$ and is contained in $\mathcal{T}_{\mathcal{C}}(\boldsymbol{q}^*_{\rightarrow})$. Considering that the measure of $S(\tilde{\boldsymbol{w}}^*_{\rightarrow})$ is non-zero, and that the residual inputs $\tilde{\boldsymbol{w}}_1, \ldots, \tilde{\boldsymbol{w}}_N$ are randomly generated in a bounded subset W of $\boldsymbol{R}^n$, the probability of choosing a sequence $\tilde{\boldsymbol{w}}_{\rightarrow}$ contained in $S(\tilde{\boldsymbol{w}}^*_{\rightarrow})$ tends to 1 as the number of iterations increases.

Let us now turn our attention to the time history $s^*(t) : [0, T] \mapsto [s_{\text{ini}}, s_{\text{fin}}]$ associated to the solution $\boldsymbol{q}^*(t)$, denoted for compactness by $s^*_{\rightarrow}$. By assumption, $s^*_{\rightarrow}$ is obtained by integrating from 0 to T a piecewise-constant sequence of $\dot{s}$

$$\dot{s}^*_{\rightarrow} = \{\dot{s}^*_1, \ldots, \dot{s}^*_{N-1}\},$$

where $\dot{s} = \dot{s}^*_k$ for $s \in [t_k, t_{k+1})$, with t_k defined as the time instant where $s = s_k$ (we have then $t_1 = 0$ and $t_N = T$). Therefore, $s^*_{\rightarrow}$ is a continuous function of $\dot{s}^*_{\rightarrow}$.

Define the *feasibility clearance* of $s^*_{\rightarrow}$ as

$$\phi = \min_{t \in [0,T]} |\boldsymbol{v}^M - \dot{\boldsymbol{q}}^*(t)|,$$

where $\dot{\boldsymbol{q}}^*(t) = \dot{s}^*(t)\,(\boldsymbol{q}^*)'(s^*(t))$. At this point, we can define a (piecewise-constant) tube in $\boldsymbol{R}$ around $s^*_{\rightarrow}$, denoted by $\mathcal{T}_{\boldsymbol{R}}(s^*_{\rightarrow})$, whose cross-section has radius ϕ. By definition, any time history that goes from 0 to T and is contained in $\mathcal{T}_{\boldsymbol{R}}(s^*_{\rightarrow})$ is feasible for $\boldsymbol{q}^*_{\rightarrow}$ (i.e., satisfies condition C1 of the K_TCMP_MO problem).

As before, based on the continuity of $s^*_{\rightarrow}$ with respect to $\dot{s}^*_{\rightarrow}$, there exists a collection of neighborhoods in $\boldsymbol{R}$

$$S(\dot{s}^*_{\rightarrow}) = \{S(\dot{s}^*_1), \ldots, S(\dot{s}^*_{N-1})\},$$

such that any choice of $\dot{s}_{\rightarrow}$ in $S(\dot{s}^*_{\rightarrow})$ (i.e., of $\dot{s}_1$ in $S(\dot{s}^*_1)$, of $\dot{s}_2$ in $S(\dot{s}^*_2)$, and so on) produces a time history $s_{\rightarrow}$ that goes from 0 to T and is contained in $\mathcal{T}_{\boldsymbol{R}}(s^*_{\rightarrow})$. Considering that the measure of $S(\dot{s}^*_{\rightarrow})$ is non-zero, and that the $\dot{s}^*_1, \ldots, \dot{s}^*_{N-1}$ are randomly generated in a bounded subset of $\boldsymbol{R}$, the probability of generating a sequence $\dot{s}_{\rightarrow}$ contained in $S(\dot{s}^*_{\rightarrow})$ tends to 1 as the number of iterations increases. Taken together with the previous conclusion on the probability of generating a suitable path, this concludes the proof for the K_TCMP_MO planner.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

The same proof applies to the non-holonomic version of the K_TCMP_MO planner as well as to the D_TCMP_MO planner. In the non-holonomic case, the only difference is that the solution path is obtained by integration of (13), whereas in the dynamic case both the solution time history and path are obtained by integrating second-order differential equations, respectively, (19) and (20). In both cases, however, it is straightforward to replicate the above arguments made for the K_TCMP_MO planner so as to prove probabilistic completeness. □

6. Exploration–Exploitation

Both K_TCMP_MO and D_TCMP_MO are randomized planners. In particular, as shown in the previous section, a random choice of the residual input vector (respectively, $\tilde{\boldsymbol{w}}$ for K_TCMP_MO and $\tilde{z}$ for D_TCMP_MO) is essential for exploring the search space and ultimately guaranteeing probabilistic completeness.

Pure random exploration, however, may prove inefficient or produce unsatisfactory motions. One possibility is to alternate exploration with *exploitation*, i.e., a phase where the residual input is chosen deterministically with the objective of minimizing a certain state-dependent cost function H. This function may either encode a heuristic for increasing the efficiency of the search, or some desirable feature of the solution. In the kinematic case, H will be a function of $\boldsymbol{q}$, whereas in the dynamic case it will depend on $(\boldsymbol{q}, \dot{\boldsymbol{q}})$.

A simple way to balance exploration and exploitation is the following. Define a threshold $\eta \in (0, 1)$. Whenever a residual input vector must be chosen (Step 2 of both planners), generate a random number $\rho \in [0, 1]$: if $\rho \geq \eta$, perform exploration as prescribed; otherwise perform exploitation. Clearly, increasing η will emphasize the role of exploitation in the planner.

For the K_TCMP_MO planner, exploitation is realized by the following choice for the residual input vector:

$$\tilde{\boldsymbol{w}} = -k_h \nabla_{\boldsymbol{q}} H(\boldsymbol{q}), \tag{23}$$

where k_h is a positive gain.

In the presence of non-holonomic constraints, the negative gradient of the cost function does not represent a feasible generalized velocity in general. Following ref. [24], we can write

$$H' = \frac{\partial H(\boldsymbol{q})}{\partial \boldsymbol{q}} \boldsymbol{q}' = \nabla_{\boldsymbol{q}}^T H(\boldsymbol{q}) \begin{pmatrix} \boldsymbol{G}(\boldsymbol{q}_c) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{n_u} \end{pmatrix} \tilde{\boldsymbol{v}},$$

where we have used (11). Therefore, the choice of $\tilde{\boldsymbol{w}}$ which realizes the best local improvement of $H(\boldsymbol{q})$ is

$$\tilde{\boldsymbol{w}}_{\text{nh}} = -k_h \begin{pmatrix} \boldsymbol{G}^T(\boldsymbol{q}_c) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{n_u} \end{pmatrix} \nabla_{\boldsymbol{q}} H(\boldsymbol{q}). \tag{24}$$

Finally, for the D_TCMP_MO planner, exploitation is realized by following choice for the residual input vector:

$$\tilde{z} = -k_h \nabla_{\dot{\boldsymbol{q}}} H(\boldsymbol{q}, \dot{\boldsymbol{q}}). \tag{25}$$

Note that probabilistic completeness is still ensured with the exploration–exploitation approach, both in the kinematic and the dynamic case. In fact, the presence of occasional exploitation phases does not alter the fact that the planner, thanks to the exploration phase, is asymptotically guaranteed to generate an appropriate combination of configuration space path and time history.

7. Planning Results

The proposed planners were implemented as C++ plugins in V-REP on a 64-bit Intel Core i7-2600K CPU running at 3.40 GHz. In particular, we present results obtained by the versions with guaranteed satisfaction of the input bounds: i.e., with automatic computation of $b_{\max}$ for K_TCMP_MO and of $c_{\max}$ for D_TCMP_MO.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

Table I. Planner parameters.

Scenario	Planner	N	r	k_p	k_d	η	α	k_h	δs	δt
1	K_TCMP_MO	11	5	10	–	0.3	1000%	10	0.002	–
2	D_TCMP_MO	11	5	100	20	0	600%	–	–	0.005
3	D_TCMP_MO	11	5	100	20	0.5	100%	1	–	0.005

Table II. Planner performance.

Scenario	Execution time	Collision checks	Motion duration	Vertexes	Mean task error
1	36.5 s	94,944	31.81 s	526	0.11 mm
2	118.5 s	298,523	6.11 s	382	0.25 mm
3	71.5 s	18,709	3.09 s	78	0.11 mm

We consider three different planning scenarios with moving obstacles. The first refers to the application of the K_TCMP_MO planner to a system of non-holonomic mobile robots. The second and third scenario involve the application of the D_TCMP_MO planner to a fixed-base manipulator. All the results are also shown in the accompanying video.

Table I reports the planner and the parameters used in each scenario. In particular,

- N is the number of sample points (equispaced in s) taken on the assigned task path;
- r is the number of random values of the residual input vector ($\tilde{\boldsymbol{w}}$ for K_TCMP_MO and $\tilde{\boldsymbol{a}}$ for D_TCMP_MO) generated by the planners at Step 2;
- k_p, k_d are the task error gains used in (8) and (20);
- η is the threshold used for implementing the exploration–exploitation mechanism of Section 6 (note that $\eta = 0$ corresponds to pure random exploration);
- α is the maximum admissible norm of the second term (the null-space term) in the motion generation schemes (8) or (20) as a percentage of the norm of the first (the range-space term) during exploration phases[(5)];
- k_h is the gain used for defining the residual input during exploitation phases, see Eqs. (23)–(25);
- δs is the stepsize used by the K_TCMP_MO planner for reconstructing configuration space paths via Euler integration in s;
- δt is the stepsize used by the D_TCMP_MO planner for reconstructing configuration space trajectories via Euler integration in t.

Table II collects the most significant performance data in the three scenarios. Since the planners are randomized, these data have been averaged over 10 runs.

Scenario 1: Kinematic planning for a system of non-holonomic mobile robots.
In the first scenario (see Fig. 4), we consider a system of four identical Kephera III robots modeled as unicycles. The configuration vector is $\boldsymbol{q} = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_4)$, where $\boldsymbol{q}_i = (x_i, y_i, \theta_i)$ is the configuration of the ith robot expressed in terms of its Cartesian position x_i, y_i, and orientation θ_i. The kinematic model of the system is in the form (10) with $n = n_c = 12$ and $n_u = 0$; in particular,

$$\dot{\boldsymbol{q}} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{v}, \quad \text{where } \boldsymbol{G}(\boldsymbol{q}) = \text{diag}\{\boldsymbol{G}_i(\boldsymbol{q}_i), i = 1, \ldots, 4\} \text{ and } \boldsymbol{G}_i(\boldsymbol{q}_i) = \begin{pmatrix} \cos\theta_i & 0 \\ \sin\theta_i & 0 \\ 0 & 1 \end{pmatrix}.$$

The input vector is $\boldsymbol{v} = (v_1, \omega_1, \ldots, v_4, \omega_4)$, where v_i, ω_i are the driving and steering velocities of the ith robot ($p = 8$). The bounds on these velocities, taken from the official documentation, are respectively, 0.5 m/s and 90°/s.

[(5)]This bound induces a bound on the norm of the random residual input ($\tilde{\boldsymbol{w}}$ or $\tilde{\boldsymbol{z}}$), which will then take values in a bounded subset (W or Z) of $\boldsymbol{R}^n$ as postulated in Step 2 of both planners and required by the proof of probabilistic completeness in Section 5.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

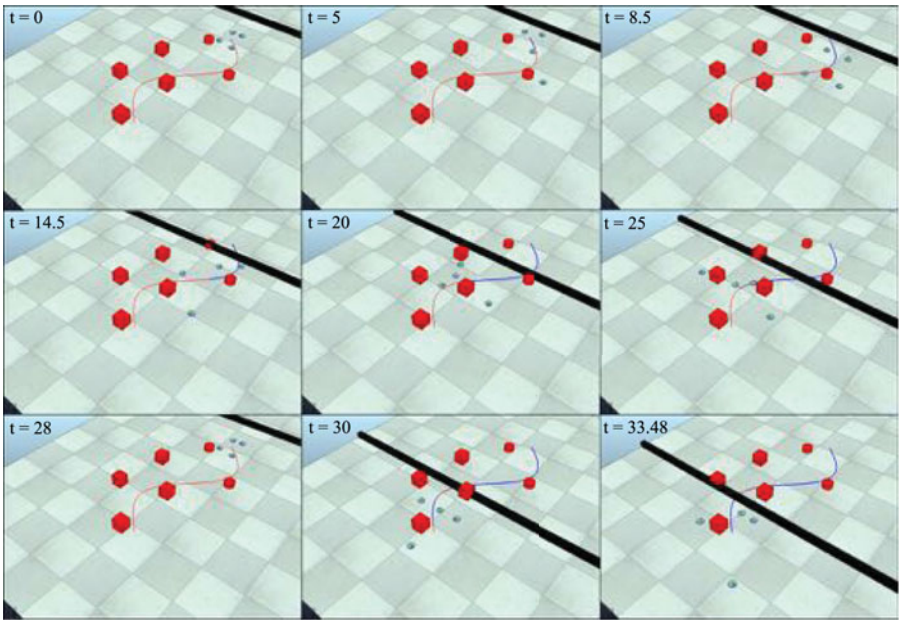


Fig. 4. Scenario 1: Four mobile robots must move so as to keep their centroid on a sinusoidal path (red) while avoiding fixed obstacles (red boxes) as well as one moving obstacle (black bar). Snapshots are taken from a solution computed by the K_TCMP_MO planner using exploration–exploitation ($\eta = 0.3$). In each snapshot, the portion of the task path traveled up to that time is shown in blue.

The task variables are the Cartesian coordinates of the centroid of the formation

$$\boldsymbol{y} = \frac{1}{4}\begin{pmatrix} \sum_{i=1}^{4} x_i \\ \sum_{j=1}^{4} y_i \end{pmatrix},$$

so that the degree of redundancy is $8 - 2 = 6$. Computing the corresponding Jacobian and using (12) provides the non-holonomic task Jacobian for this case:

$$\boldsymbol{J}_{\mathrm{nh}}(\boldsymbol{q}) = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{G}(\boldsymbol{q}) = \frac{1}{4}\begin{pmatrix} c_1 & 0 & c_2 & 0 & c_3 & 0 & c_4 & 0 \\ s_1 & 0 & s_2 & 0 & s_3 & 0 & s_4 & 0 \end{pmatrix}.$$

The assigned path for the centroid is a sine wave of length 3.70 m.

During the motion the robots must avoid six fixed obstacles and one wide moving obstacle that sweeps the entire workspace; obviously, the robots must also avoid each other. Being this a non-holonomic system, we applied the K_TCMP_MO planner in the modified version of Section 3.4, with the parameter settings shown in the first row of Table I.

Since the task only concerns the formation centroid, the robots may in principle drift far away from the assigned path. To avoid this, we have used the exploration–exploitation mechanism of Section 6. In particular, the residual input during exploitation phases is obtained by (23) using as cost function the variance of the formation, defined as the sum of the squared distances of the robots from the centroid:

$$H(\boldsymbol{q}) = \sum_{i=1}^{4}(x_i - \sum_{j=1}^{4} x_j/4)^2 + (y_i - \sum_{j=1}^{4} y_j/4)^2.$$

Figure 4 shows some snapshots from a solution computed by the planner (see the accompanying video for a clip). The robots realize the assigned task while effectively avoiding all collisions; at the same time, thanks to the exploration–exploitation mechanism, they keep a relatively tight formation around the centroid. As shown by Fig. 5, there are no motion inversions in this solution, due to the size of the moving obstacle (the bar) which sweeps the entire workspace and forces the robots to escape in the half plane in front of it.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

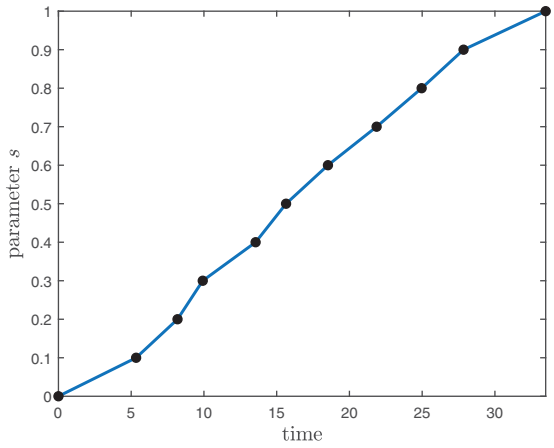


Fig. 5. Scenario 1: Time history along the solution.

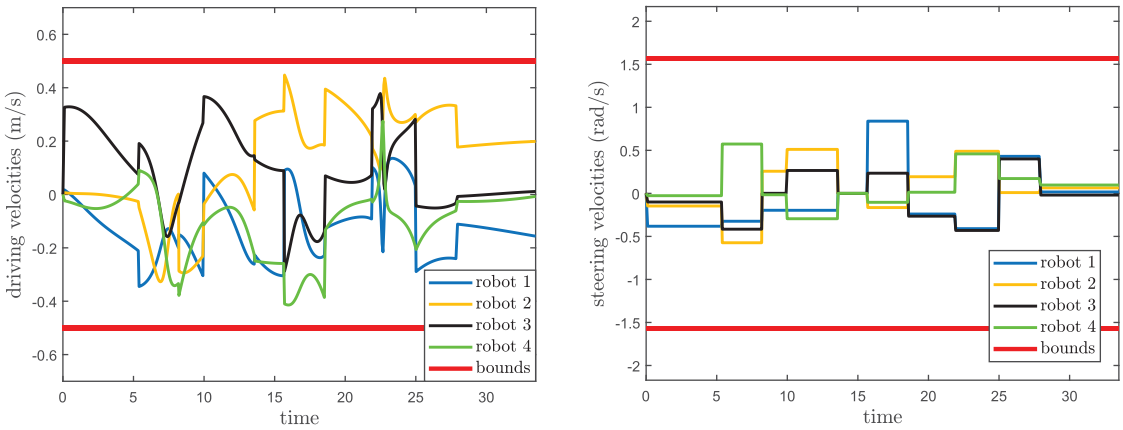


Fig. 6. Scenario 1: Velocity inputs required by the solution.

The velocity inputs for all robots, shown in Fig. 6, are always within the imposed bounds. The piecewise-constant profile of the steering velocities is due to the special structure of matrix $\boldsymbol{J}_{\text{nh}}$, which implies that $\boldsymbol{J}_{\text{nh}}^{\dagger}$ has null rows in correspondence of the angular velocities and $(\boldsymbol{I}_8 - \boldsymbol{J}^{\dagger}_{\text{nh}}\boldsymbol{J}_{\text{nh}})$ has only 0/1 elements in the same rows. As a consequence of these two facts, steering velocities inputs computed through (13) using piecewise-constant residual inputs will be piecewise-constant themselves.

The first row of Table II summarizes the typical performance of the K_TCMP_MO planner in this scenario. Remember that these are averaged data over 10 runs and therefore they do not correspond exactly to the particular solution so far discussed. Note the very low value of the mean task error, which is continuously computed over the whole robot trajectory.

To illustrate the benefits of the exploration-exploitation mechanism, we present in Fig. 7 a solution computed by the K_TCMP_MO planner using only exploration (this corresponds to setting $\eta = 0$). As expected, the robots drift far away from the path assigned to the centroid; this is confirmed by the comparison of variance in Fig. 8. On the other hand, the planner took 8.9 s to compute this solution, less than one-fourth of the time needed with $\eta = 0.3$; this is due to the fact that the exploitation, which was aimed at minimizing a quantity (the variance) unrelated to exploration, actually got in the way of search efficiency. Still, considering that ours is an off-line planner, the execution time is reasonable in both cases.

While piecewise-continuous velocity inputs are usually admissible in mobile robots equipped with low-level servo loops, we would nonetheless like to conclude the discussion of this scenario with a comment about the possibility of obtaining velocity signals that are continuous over time. There

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

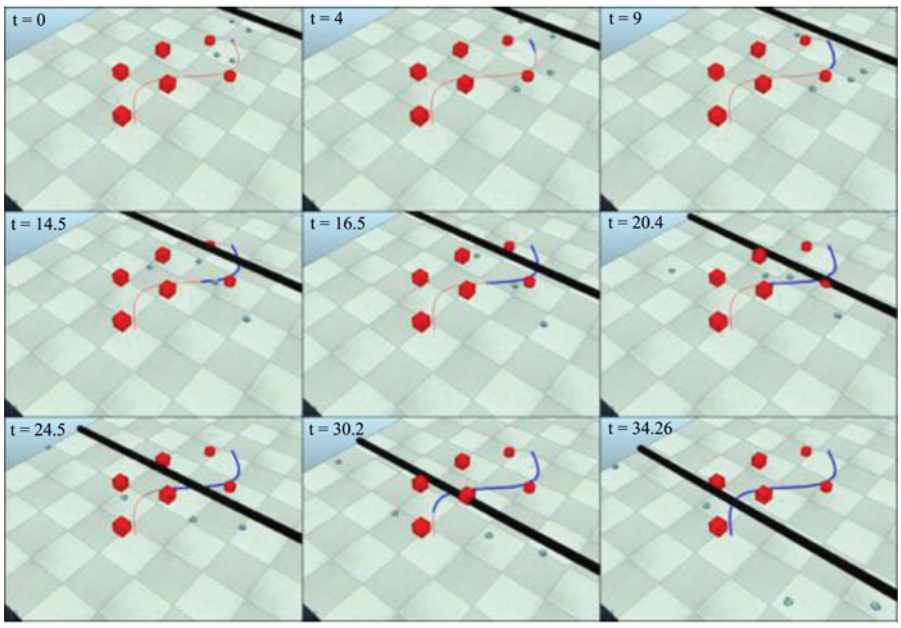


Fig. 7. Scenario 1: A different solution computed by the K_TCMP_MO planner using pure exploration ($\eta = 0$).

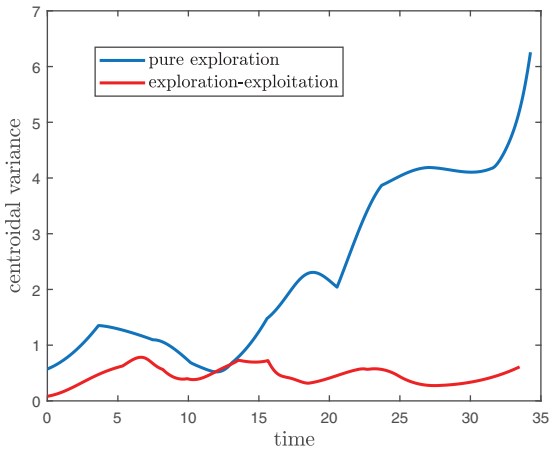


Fig. 8. Scenario 1: Centroidal variance for the two solutions.

are two fully viable options to achieve this within the proposed framework. The first is to represent each unicycle by its dynamic model, where the control inputs are accelerations, rather than by its kinematic model; at this point, using the D_TCMP_MO planner, one will obtain piecewise-constant accelerations and, therefore, continuous velocities. The second option is to keep the kinematic model, with the control inputs as velocities, and to choose the residual input vector $\tilde{\boldsymbol{w}}$ in (8) with a *piecewise-linear* (rather than piecewise-constant) profile over s. In particular, the same value of $\tilde{\boldsymbol{w}}$ should be used when entering and exiting a vertex. A similar modification has already been proposed in ref. [25], to which we refer for details.

Scenario 2: Dynamic planning for a welding manipulator.
In the second scenario (see Fig. 9), a seven-dof KUKA LWR-IV manipulator is assigned a welding task. In particular, the welding torch mounted on the tip of the manipulator must move along a circular soldering path located on a mechanical piece, while another manipulator is moving on a pre-programmed trajectory in the same area.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

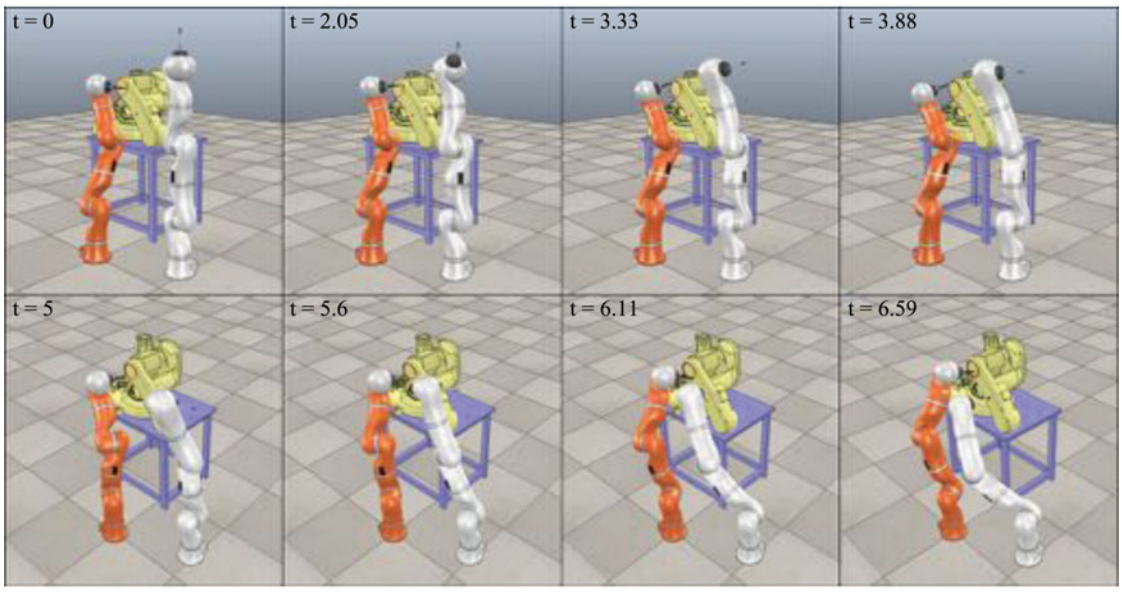


Fig. 9. Scenario 2: The orange manipulator must perform a welding task (red circle) on a mechanical piece (yellow) in the presence of another manipulator (white) which acts as a moving obstacle. Snapshots are taken from a solution computed by the D_TCMP_MO planner. In each snapshot, the portion of the task path traveled up to that time is shown in blue.

The configuration is $\boldsymbol{q} = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_6)$, where q_i is the ith joint variable (the wrist roll joint is frozen since it does not affect end-effector positioning). The model of the system is in the form (15), with bounds on joint positions, velocities, and torques derived from the official documentation of the LWR-IV. The task variables are the Cartesian coordinates of the welding torch tip, so that the degree of redundancy is $6 - 3 = 3$.

Since this is a dynamic system, we applied the D_TCMP_MO planner, with the parameter settings shown in the second row of Table I. For this scenario, we have chosen to perform pure exploration ($\eta = 0$) in view of the highly constrained nature of the problem.

Figure 9 displays some snapshots from a solution computed by the planner (see the accompanying video for a clip). The manipulator is able to realize the assigned task while avoiding all collisions. As shown by Fig. 10, also in this case, there are no motion inversions in the solution. Figure 11 confirms that the joint torques along the solution comply with the bounds, the same is true for the joint velocities, not shown here for lack of space.

The second row of Table II summarizes the typical performance of the D_TCMP_MO planner in this scenario. The longer time required to find a solution is due to the fact that the assigned task path is in contact with the surface of an obstacle, leading to a large number of motions rejected for collision. The mean task error is again very small.

Scenario 3: Dynamic planning for a manipulator moving on a cyclic path.
In the third scenario (see Fig. 12), a seven-dof KUKA LWR-IV manipulator must move its tip along a circular path, while a spherical obstacle moves back and forth on the same path. Since the robot is the same of Scenario 2, the same model and bounds apply. The degree of redundancy is again 3.

The choice of parameters for the D_TCMP_MO planner for this scenario is shown in the third row of Table I. In this case, we have chosen to perform exploration–exploitation ($\eta = 0.5$); in particular, during exploitation the residual input is computed as in (25) with the kinetic energy as cost function:

$$H(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} \dot{\boldsymbol{q}}^T \boldsymbol{B}(\boldsymbol{q}) \dot{\boldsymbol{q}}.$$

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

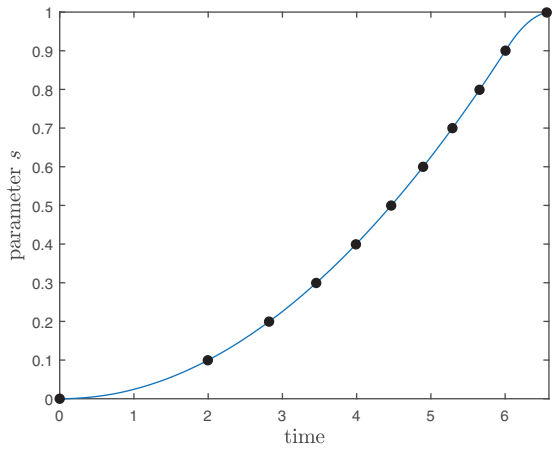


Fig. 10. Scenario 2: Time history along the solution.

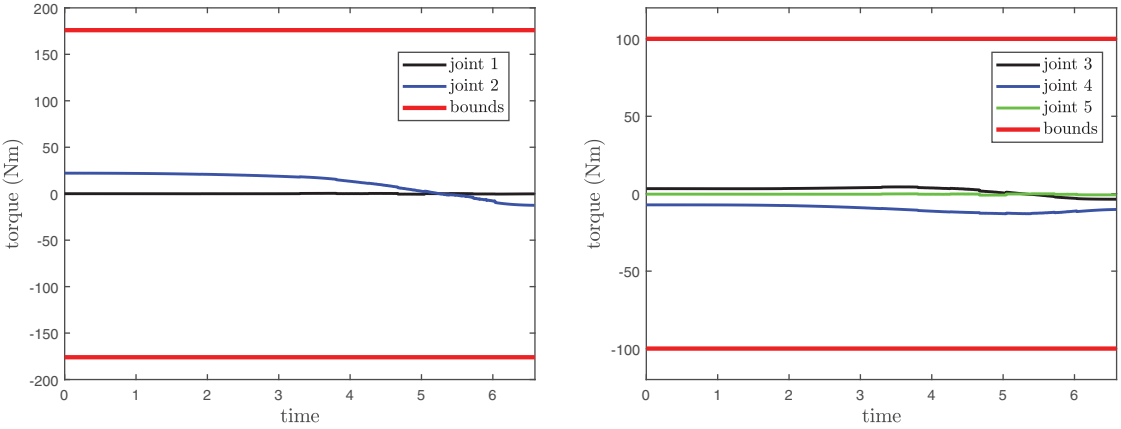


Fig. 11. Scenario 2: Torque inputs required by the solution (sixth joint torque is negligible, while the seventh joint is frozen).

The rationale here was stay away from kinematically singular configurations (quite likely to occur in view of the placement of the end-effector path, see Fig. 12) by avoiding the velocity buildup associated to them.

Figure 12 shows some snapshots from a solution computed by the planner (see the accompanying video for a clip). In order to trace the whole assigned path with its end-effector and avoid the moving obstacle at the same time, the robot performs two forward and one backward motion phases; this is confirmed by the two motion inversions in Fig. 13. The joint torques (see Fig. 14) are always below the given bounds; the same is true for the joint velocities.

A close scrutiny of this solution reveals that the backward motion of the end-effector along the path is not strictly required (stopping would have been sufficient) and could easily be eliminated in a post-processing phase (not *a priori*, because it is easy to build scenarios where obstacles move in such a way to make backward motions essential for solving the problem). This indicates that the planned motion is not optimal from a viewpoint of traveled space, consistently with the fact that our problem formulation does not include the optimization of a cost criterion. One way to improve the quality of the solution under this aspect would be to perform exploration–exploitation using a cost criterion related to the length of the configuration space path.

Averaged data on the performance of the D_TCMP_MO planner in this scenario are shown in the third row of Table II. Although there is a single obstacle to be avoided, the time needed to find a solution is non-negligible, essentially due to the elaborate temporal-spatial nature of the required motion.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

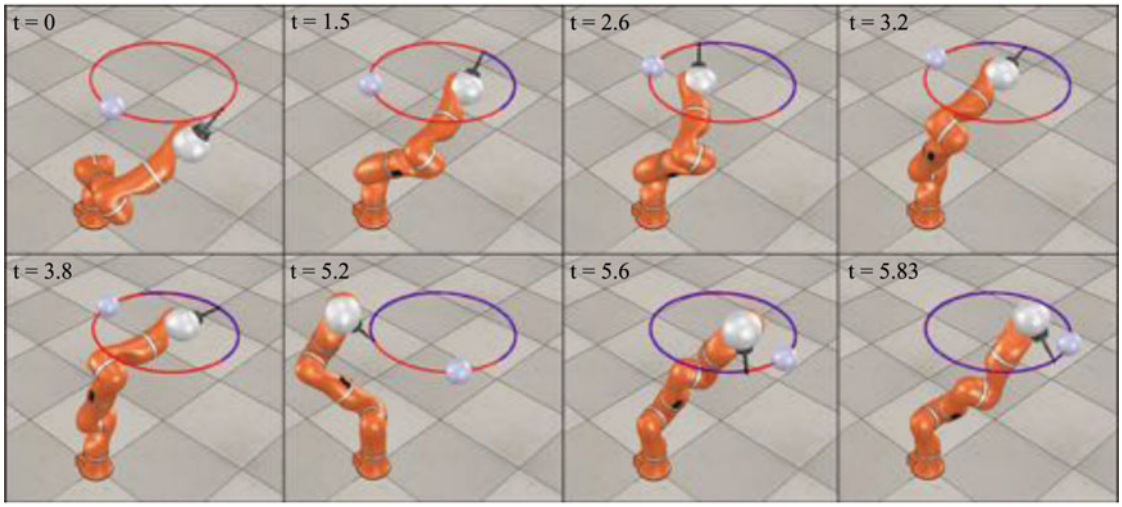


Fig. 12. Scenario 3: The manipulator must move its end effector along a circular path (in the counterclockwise direction) while avoiding an obstacle in motion on the same path. Snapshots are taken from a solution computed by the D_TCMP_MO planner. In each snapshot, the portion of the task path traveled up to that time is shown in blue.

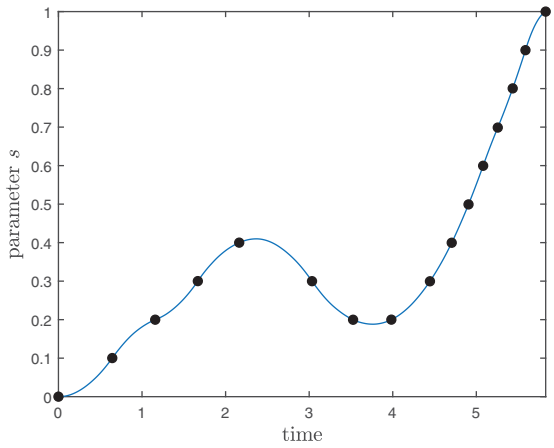


Fig. 13. Scenario 3: Time history along the solution. Note the motion inversions.

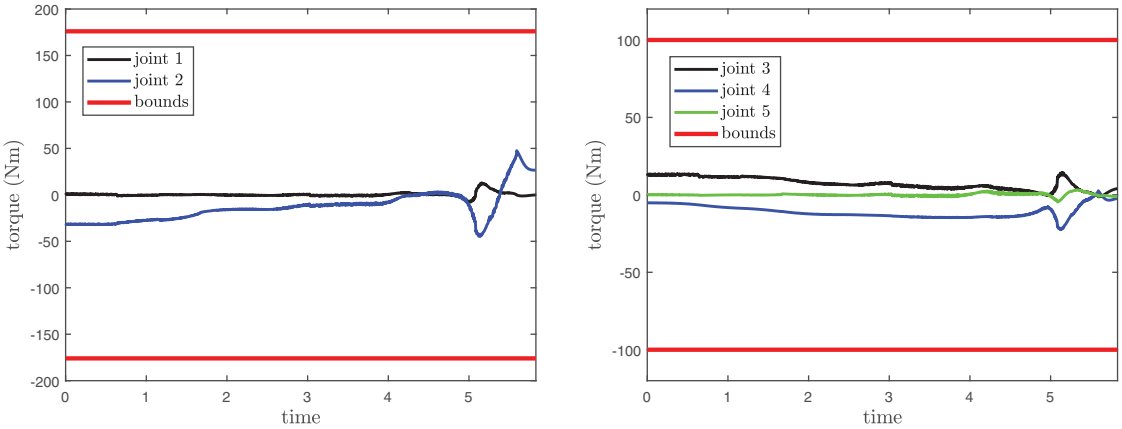


Fig. 14. Scenario 3: Torque inputs required by the solution (sixth joint torque is negligible, while the seventh joint is frozen).

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

In this case, the exploration–exploitation mechanism actually improves the efficiency of the search, because it limits the number of candidate motions that are discarded due to proximity to singular configurations (see Step 3 of the extension procedure in Section 4.3.2); in particular, the average of this number over 10 runs was 78. In comparison, use of the D_TCMP_MO planner using only exploration (i.e., setting $\eta = 0$) leads to an average of 796 discarded motions; correspondingly, the execution time was increased to 1363 s.

8. Conclusions

Consider the practically relevant situation where a robotic system is assigned a task to be executed in an environment which contains moving obstacles. Generating motions that allow the robot to execute the task, comply with its control input limitations, and are collision-free is a challenging problem whose solution must be sought in the robot state space extended with time. We describe a general planning framework which can be tailored to robots described by either kinematic or dynamic models. In both cases, the main component is a control-based scheme for producing configuration space subtrajectories along which the task constraints are continuously satisfied. The geometric motion and the time history along the subtrajectory are generated separately in order to guarantee the feasibility of the latter and at the same time make the scheme intrinsically more flexible. A randomized algorithm then explores the search space by repeatedly invoking the motion generation scheme and checking the produced subtrajectories for collisions. The proposed framework is shown to provide a probabilistically complete planner both in the kinematic and the dynamic case. Modified versions of the planners based on the exploration–exploitation approach can also be devised to improve search efficiency or optimize a performance criterion along the solution. We present planning results in various scenarios involving both mobile robots and fixed-based manipulators to show the effectiveness of the proposed method.

We are currently working to extend the proposed approach to the case of underactuated robots. The challenge of underactuation, which is naturally present in many advanced robotic mechanisms (e.g., robotic hands, humanoids, and UAVs), is obviously that arbitrary generalized accelerations cannot be produced. Preliminary results in this direction have been presented in ref. [26]. Another extension we are currently considering is the development of an anytime version of the proposed planner, allowing to relax the assumption of fully known obstacle motion and using instead short-term predictions computed on the basis of sensory information.

Supplementary Material

To view supplementary material for this article, please visit https://doi.org/10.1017/S0263574718001182.

References

1. J. Reif and M. Sharir, "Motion planning in the presence of moving obstacles," *J. ACM* **41**(4), 764–790 (1994).
2. K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Robot. Res.* **5**(3), 72–89 (1986).
3. M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica* **2**(4), 477–521 (1987).
4. T. Fraichard, "Dynamic trajectory planning with dynamic constraints: A 'state-time space' approach," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (1993) pp. 1393–1400.
5. P. Fiorini and Z. Shiller, "Time optimal trajectory planning in dynamic environments," *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, MN (1996) pp. 1553–1558.
6. P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.* **17**(7), 760–772 (1998).
7. Z. Shiller, F. Large and S. Sekhavat, "Motion planning in dynamic environments: Obstacles moving along arbitrary trajectories," *Proceedings of the IEEE International Conference on Robotics and Automation*, Seoul, Korea (2001) pp. 3716–3721.
8. B. Donald, P. Xavier, J. Canny and J. Reif, "Kinodynamic motion planning," *J. ACM* **40**(5), 1048–1066 (1993).

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

9. D. Hsu, R. Kindel, J. C. Latombe and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.* **21**(3), 233–255 (2002).
10. A. Inoue, K. Inoue and Y. Okawa, "On-line motion planning of an autonomous mobile robot to avoid multiple moving obstacles based on the prediction of their future trajectories," *J. Robot. Soc. Japan* **15**(2), 249–260 (1997).
11. T. Mercy, W. Van Loock and G. Pipeleers, "Real-time motion planning in the presence of moving obstacles," *Proceedings of the European Control Conference (ECC)* (2016) pp. 1586–1591.
12. C. Stachniss and W. Burgard, "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments," *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland (2002) pp. 508–513.
13. J. Van den Berg, D. Ferguson and J. Kuffner, "Anytime path planning and replanning in dynamic environments," *Proceedings of the IEEE International Conference on Robotics and Automation*, Orlando, FL (2006) pp. 2366–2371.
14. V. Narayanan, M. Phillips and M. Likhachev, "Anytime safe interval path planning for dynamic environments," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Algarve, Portugal (2012) pp. 4708–4715.
15. L. Kavraki, P. Svestka, J. C. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning high-dimensional configuration spacess," *IEEE Trans. Robot. Autom.* **12**(4), 566–580 (1996).
16. S. M. LaValle, Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical Report, Computer Science Dept., Iowa State University (1998).
17. D. Berenson, S. Srinivasa, D. Ferguson and J. Kuffner, "Manipulation planning on constraint manifolds," *Proceedings of the IEEE International Conference on Robotics and Automation*, Kobe, Japan (2009) pp. 625–632.
18. M. Stilman, "Global manipulation planning in robot joint space with task constraints," *IEEE Trans. Robot.* **26**(3), 576–584 (2010).
19. C. Suh, B. Kim and F. Park, "The tangent bundle RRT algorithms for constrained motion planning," *Proceedings of the 13th World Congress in Mechanism and Machine Science*, Guanajuato, Mexico (2011) pp. 1–5.
20. L. Jaillet and J-M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Trans. Robot.* **29**(1), 105–117 (2013).
21. G. Oriolo and M. Vendittelli, "A control-based approach to task-constrained motion planning," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, MO (2009) pp. 297–302.
22. M. Cefalo, G. Oriolo and M. Vendittelli, "Task-constrained motion planning with moving obstacles," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan (2013) pp. 5758–5763.
23. M. Cefalo and G. Oriolo, "Dynamically feasible task-constrained motion planning with moving obstacles," *Proceedings of the IEEE International Conference on Robotics and Automation*, Hong Kong, China (2014) pp. 2045–2050.
24. A. De Luca, G. Oriolo and P. Robuffo Giordano, "Image-based visual servoing schemes for nonholonomic mobile manipulators," *Robotica* **25**(2), 131–145 (2007).
25. G. Oriolo, M. Cefalo and M. Vendittelli, "Repeatable motion planning for redundant robots over cyclic tasks," *IEEE Trans. Robot.* **33**(5), 1170–1183 (2017).
26. M. Cefalo and G. Oriolo, "Task-constrained motion planning for underactuated robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, Seattle, WA (May 6–10, 2015) pp. 2965–2970.

9. Appendix: Planner pseudocode

A pseudocode description of the proposed planner is given in Algorithm 1. In particular, this is the K_TCMP_MO planner described in Section 3.3, with the associated procedure for tree extension described in Section 3.3.2. For simplicity, in the latter, we have assumed that only forward subpaths are generated.

For the case of non-holonomic systems considered in Section 3.4, one only needs to modify the EXTEND procedure as follows: (1) in lines 2–3, replace $\tilde{\boldsymbol{w}} \in W \subset \boldsymbol{R}^n$ with $\tilde{\boldsymbol{w}}_{\mathrm{nh}} \in W \subset \boldsymbol{R}^p$ (2) in line 3, paths should be produced by integration of (10)–(13). Finally, starting from Algorithm 1, it is easy to write the pseudocode description for the D_TCMP_MO planner of Section 4.3.

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press

Algorithm 1: K_TCMP_MO planner

input: $\boldsymbol{q}_{\text{ini}}$; $\boldsymbol{y}_d(s)$, $s \in [s_{\text{ini}}, s_{\text{fin}}]$
output: $\boldsymbol{q}(t)$, $t \in [0, T]$

root the tree $\mathcal{T}$ at $(\boldsymbol{q}_{\text{ini}}, 0)$;
repeat
  $i \leftarrow i + 1$;
  select a random sample $\boldsymbol{y}_{\text{rand}}$ from the predefined sequence $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$;
  compute an inverse solution $\boldsymbol{q}_{\text{rand}}$ associated to $\boldsymbol{y}_{\text{rand}}$;
  select a random time instant $t_{\text{rand}} \in [0, t_{\max}]$;
  find the vertex $(\boldsymbol{q}_{\text{near}}, t_{\text{near}})$ in $\mathcal{T}$ that is closest to $(\boldsymbol{q}_{\text{rand}}, t_{\text{rand}})$;
  get the parameter value s_k associated with $\boldsymbol{q}_{\text{near}}$;
  $[\boldsymbol{q}_{\text{new}}, t_{\text{new}}, s_{k+1}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}] \leftarrow \text{Extend}(\boldsymbol{q}_{\text{near}}, t_{\text{near}}, s_k)$;
  if $\boldsymbol{q}_{\text{new}} \neq \emptyset$ **then**
    add vertex $(\boldsymbol{q}_{\text{new}}, t_{\text{new}})$ and edge $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ to $\mathcal{T}$;
until $s_{k+1} = s_N$ **or** $i = \text{MAX_IT}$;

Procedure Extend($\boldsymbol{q}_{\text{near}}, t_{\text{near}}, s_k$)

input: $\boldsymbol{q}_{\text{near}}, t_{\text{near}}, s_k$
parameters: $b_{\max}, r, W, k_p, k_d, \alpha, \delta s$
output: $\boldsymbol{q}_{\text{new}}, t_{\text{new}}, s_{k+1}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$

select a random value of $\dot{s}$ in $[0, b_{\max}]$, and compute the corresponding time history from s_k to s_{k+1};
randomly choose r values for $\tilde{\boldsymbol{w}}$ in the bounded set $W \subset \boldsymbol{R}^n$;
for each value of $\tilde{\boldsymbol{w}}$, generate the corresponding subpath from $\mathcal{L}_k$ to $\mathcal{L}_{k+1}$ by integrating (5)–(8) from s_k to s_{k+1};
retain only the subpath terminating at the configuration closest to $\boldsymbol{q}_{\text{rand}}$, called $\boldsymbol{q}_{\text{new}}$;
reconstruct the corresponding subtrajectory $\overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}$ by adding the time history;
if collision **or** joint position/velocity limit violation **then**
  return $[\emptyset, \emptyset, \emptyset]$;
else
  return $[\boldsymbol{q}_{\text{new}}, t_{\text{new}}, s_{k+1}, \overline{\boldsymbol{q}_{\text{near}}\boldsymbol{q}_{\text{new}}}]$;

https://doi.org/10.1017/S0263574718001182 Published online by Cambridge University Press