Autonomous Robots (2019) 43:631–648
https://doi.org/10.1007/s10514-018-9728-3

Goal state driven trajectory optimization

Avishai Sintov[1,2]

Received: 27 June 2016 / Accepted: 2 April 2018 / Published online: 16 April 2018

Abstract
Many applications demand a dynamical system to reach a goal state under kinematic and dynamic (i.e., kinodynamic) constraints. Moreover, industrial robots perform such motions over and over again and therefore demand efficiency, i.e., optimal motion. In many applications, the initial state may not be constrained and can be taken as an additional variable for optimization. The semi-stochastic kinodynamic planning (SKIP) algorithm presented in this paper is a novel method for trajectory optimization of a fully actuated dynamic system to reach a goal state under kinodynamic constraints. The basic principle of the algorithm is the parameterization of the motion trajectory to a vector in a high-dimensional space. The kinematic and dynamic constraints are formulated in terms of time and the trajectory parameters vector. That is, the constraints define a time-varying domain in the high dimensional parameters space. We propose a semi stochastic technique that finds a feasible set of parameters satisfying the constraints within the time interval dedicated to task completion. The algorithm chooses the optimal solution based on a given cost function. Statistical analysis shows the probability to find a solution if one exists. For simulations, we found a time-optimal trajectory for a 6R manipulator to hit a disk in a desired state.

Keywords Motion planning · Kinodynamic constraints · Trajectory optimization

1 Introduction

For the robotic system to accomplish its designated tasks, it must be able to generate a feasible motion. Usually, the motion is along a trajectory from an initial state (generalization vector of the position and velocity) to a desired goal state under kinematic and dynamic constraints, denoted in short as kinodynamic constraints. Examples for such motions are robotic arm manipulation tasks (Kim et al. 2013), motion of autonomous vehicles (Heo and Chung 2013), and UAV's (Motonaka et al. 2013). Feasible motions are those that obey the mechanical and environmental constraints of the system, i.e., satisfy the kinodynamic constraints (Donald et al. 1993; Pham et al. 2013). Mechanical constraints are physical limitations of the system such as joints limits or maximum and minimum bounds of the joints' velocities and torques. Environmental constraints are generally static or dynamic obstacles in the workspace of the system that limit its motion in some states. To acquire a feasible motion of the dynamic system, a planning algorithm is required to generate a trajectory that allows motion satisfying all constraints.

An unconstrained robot may perform motion with excessively high velocities and torques to reach a desired state. However, when obvious limitations on actuators' position, velocity and torques are applied, and additional workspace obstacles are present, a feasible collision-free trajectory must be found. Moreover, in industrial processes, achieving efficient manufacturing is essential for maximizing the productivity (Verscheure et al. 2009). An industrial robot will perform a routine trajectory over and over again. Therefore, an optimal motion that minimizes, for example, time, energy, torques/forces, path length, or velocities, must be applied. Hence, we seek an optimal collision-free trajectory that satisfies the kinodynamic constraints.

A general trajectory optimization for a given task requires start and goal states. The start state is usually the current pose of the robot while the goal state is defined by the desired task. In many applications, the goal state is the parameter that matters. For example, throwing an object from a desired

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10514-018-9728-3) contains supplementary material, which is available to authorized users.

✉ Avishai Sintov
asintov@illinois.edu

[1] Coordinated Science Laboratory, University of Illinois as Urbana-Champaign, Urbana, IL 61801, USA

[2] Department of Mechanical Engineering, Ben-Gurion University of the Negev, 84105 Beer-Sheva, Israel

position and velocity, hitting an object in a specific force, regrasping an object, soft catching a free-flying object, etc. When a specific initial state is not a necessary constraint as in the examples above, the optimal trajectory problem has additional free parameters for optimization. Thus, the number of possible solutions will increase and the acquisition of better ones is possible. Therefore, in this work we propose to apply a trajectory-optimization algorithm that also optimizes the initial state of the robot. This planning problem is given the term: the *Goal State driven Trajectory Optimization Problem* (GSTOP).

We propose to divide the planning into two parts: plan a trajectory to the goal while optimizing the initial state as well, and plan an administrative path to that initial state from the current pose of the robot. The main trajectory will satisfy the kinodynamic constraints and will be optimized. On the contrary, the planning of the administrative path can rely solely on kinematic constraints. A prominent real-life example of this approach can be taken from the world of baseball where a batter tries to hit a thrown ball. The batter will take an administrative motion to position the bat above his shoulder. Then, he will execute a somewhat optimal trajectory for the bat to hit the ball flying toward him. It would not make sense nor be optimal for the batter to start the hitting motion from an arbitrary pose. Another example is in-hand regrasping motions where a robotic arm releases the object and catches it in a different relative pose (Sintov and Shapiro 2017). The choice of the release pose is essential for a successful completion of the task and should thus be optimized. Thus, the motion is no longer constrained by the initial pose and the administrative path enables the choice of the right starting state. We also note that this approach could be used to improve the performance of any kinodynamic system with an existing trajectory previously found for some initial state (excluding pick-and-place operations where the initial state cannot be changed). Thus, by throwing away the initial state and replanning with the proposed approach, one could find a new and better initial state that reduces the operational cost. In this paper we disregard the planning of the administrative path where many planners exist and focus on the GSTOP.

We present a novel algorithm termed Semi-Stochastic KInodynamic Planning (SKIP) as a method for finding an optimal solution for the motion planning problem taking the kinodynamic constraints into account. Applications of the algorithm for object throwing and regrasping have been presented by the authors in Sintov and Shapiro (2015) and Sintov and Shapiro (2017), respectively. However, in this paper we present a general and extended algorithm for any dynamic system. We focus on finding a feasible and optimal trajectory in some finite time while only the goal state is given. We propose a different parameterization of any trajectory for a dynamical system. In addition, as opposed to the previous work, here we also consider equality constraints. The proposed method can generally optimize both initial pose and velocity. We note however that for practical applications, the administrative path should be simple and move the robot to complete stop at the required initial pose. Therefore, without loss of generality, we only focus on the optimization of the initial pose and set the velocity to zero.

The key component of the algorithm is parameterizing an analytic trajectory function with redundant parameters to serve along with the goal time as free parameters for the optimization. We formulate the kinodynamic constraints of the problem in the free parameters space. The formulated set of constraints defines the Time-Varying Constraint (TVC) problem and an easy-to-use numerical method is proposed. The numerical method is a semi-stochastic algorithm, and a statistical analysis is presented to calculate the probability to find a solution if one exists. In general, similar to other sampling-based approaches, we show that the probability to find a solution if one exists, approaches one as the number of generated random points increases to infinity. The statistical analysis provides an automatic parameter selection for determining certain parameters for the algorithm. An important advantage of this method is the ability to calculate the probability to find a solution based on the allowed computational run-time.

The paper is organized as follows. Related work is presented in Sect. 2. Section 3 defines the motion planning problem with its constraints. In Sect. 4 some preliminary background notions are presented. In Sect. 5 we formulate the kinodynamic constraints in a specific manner based on mechanical limitations and the control method. The formulated constraints define the TVC problem, which is solved in Sect. 6. Section 7 performs statistical and complexity analysis of the algorithm. In Sect. 8 we present simulations of a 6R manipulator motion while avoiding dynamic obstacles to show the feasibility of the algorithm. Conclusions and future work are presented in Sect. 9.

2 Related work

The motion planning problem under kinodynamic constraints has been under extensive research in the past three decades. During that time, it has been shown that a complete algorithm for only the kinematic planning problem is sufficiently difficult in terms of computational costs (Schwartz and Sharir 1983; Brooks and Lozano-Perez 1983; Canny 1988; Reif 1979). Therefore, it is hard to implement them in practical applications. Some work applied some assumptions to reduce complexity; in Van der Stappen et al. (1998) the algorithm assumes low density obstacles and therefore lower complexity bounds exist, or in Van der Stappen et al. (1993) the assumption is that the robot is relatively small compared to the obstacles.

Finding an optimal trajectory under kinodynamic constraints has two main approaches: trajectory optimization and optimal control. Distinction between the two is well defined in Rao (2009). While in optimal control we desire an input function to acquire an optimal trajectory, in trajectory optimization we seek for static parameters that define the optimal trajectory. Although optimal control is a well-established method for finding an optimal trajectory, its capabilities to handle obstacle avoidance is rather inadequate (Zucker et al. 2013). Moreover, the solution of an optimal control demands an initial state of the system, which invalidates its usage for GSTOP. In addition, the work presented in this paper is closer to trajectory optimization and therefore, we focus on related work in that field. Khatib (1986) first introduced artificial potential fields for finding a collision-free trajectory. Rimon and Koditschek (1992) later extended the method with navigation functions free of local minima. One of the recent and prominent methods is the CHOMP (Ratliff et al. 2009; Zucker et al. 2013) and its variants (Dragan et al. 2011; Kalakrishnan et al. 2011; Byravan et al. 2014; Park et al. 2012). CHOMP is a functional gradient descent method that continuously refines the trajectory toward collision freeness. More prominent methods are the TrajOpt (Schulman et al. 2013) and the shooting-method (Keller 1976). An extended survey on trajectory optimization methods can be found in Rao (2014).

In the motivation to find more practical algorithms, probabilistic sampling methods have become common. They provide easy to implement methods with relatively low complexity, good results, and efficiency planning in high-dimensional configuration spaces. Moreover, they do not require an analytical representation of the obstacles. The most common probabilistic methods are the Probabilistic Roadmaps (PRM) and the Rapidly-exploring Random Trees (RRT). PRM (Kavraki et al. 1996; Hsu et al. 2002; Song and Amato 2001; Clark 2005; Denny et al. 2013) is a kinematic planning method that generates random states in the state space and constructs a graph by connecting the random states with collision-free trajectories. RRT was first introduced in Lavalle (1998) and LaValle and Kuffner (1999) as a randomized approach for kinodynamic planning. It incrementally builds a search tree in the state space while integrating the control inputs to ensure that the kinodynamic constraints are satisfied. Some other extensions of the RRT are the CL-RRT (Luders et al. 2010), and TB-RRT (Sintov and Shapiro 2014). One of the drawbacks of PRM and RRT algorithms is that they provide an arbitrary solution rather than an optimal one. Karaman and Frazzoli introduced the PRM* and RRT* (Karaman and Frazzoli 2011), which use the notion of PRM and RRT, respectively, but ensures an optimal solution with probability one as the number of random points approaches infinity. However, both extended methods can be applied to systems where motion between any two states can be made on a straight line. Hence, differential constraints of the system cannot be applied. The work in Webb and van den Berg (2013) extended the RRT* for systems with linear differential constraints, this by optimally connecting any pair of states in the tree. All of the above mentioned methods demand an initial state for the motion. Hence, they cannot be applied for the GSTOP discussed in this paper.

3 Problem formulation

In this section we define the dynamic motion planning problem including the constraints and assumptions.

3.1 Given system and environment

Let $\mathcal{Q} \subseteq \mathbb{R}^n$ be the configuration space of a fully actuated dynamic system with n degrees of freedom and let $\mathcal{U} \subseteq \mathbb{R}^n$ be its set of all possible force/torque inputs. Moreover, let $\Upsilon \subset \mathbb{R}_{\geq 0}$ be the time space of the problem. The equations of motion of the dynamic system are given

$$M(\phi)\ddot{\phi} + C(\phi, \dot{\phi})\dot{\phi} + G(\phi) = \mathbf{u}, \tag{1}$$

where $\phi(t) \in \mathcal{Q}$ is the configuration of the system at time $t \in \Upsilon$, M is an $n \times n$ inertia matrix, C is an $n \times n$ matrix of centrifugal and Coriolis acceleration terms, G is an $n \times 1$ vector of gravitational effects, and $\mathbf{u}(t) \in \mathcal{U}$ is a vector of control input forces and torques to the actuated degrees of freedom (DOF) at time $t \in \Upsilon$.

Let $\mathcal{Q}_{res} \subset \mathcal{Q}$ be a restricted subspace of the configuration space defined by physical limitations of the system's DOF, e.g., joint mechanical limitations in a robotic arm. In addition, the environment of the system comprises dynamic obstacles. Formally, the obstacle region is a restricted time-varying subset denoted by $\mathcal{QT}_{obs} \subset \mathcal{Q} \times \Upsilon$. Therefore, we define a projection map from such space to the configuration space as follows.

Definition 1 Let the map $\Pi_t : \mathcal{Q} \times \Upsilon \rightarrow \mathcal{Q}$ be a projection map of the time-varying configuration space to the configuration space at time t.

With this system and its environment we can now define the bounding constraints.

3.2 Constraints

The motion of the dynamic system is limited under the following constraints:

1. The dynamic system is constrained to be in the allowed configuration space $\mathcal{Q}^t_{free} = \mathcal{Q} \setminus (\mathcal{Q}_{res} \cup \mathcal{Q}^t_{obs})$ at all times t, where $\mathcal{Q}^t_{free}$ is the allowed configuration $\mathcal{Q}_{free} \subseteq \mathcal{Q}$ at time t and $\mathcal{Q}^t_{obs} = \Pi_t \circ \mathcal{QT}_{obs}$.

2. The allowed control inputs are defined by the abilities of the actuators and given by $\mathcal{U}_{al} \subseteq \mathcal{U}$.
3. Let $\mathcal{V} \subseteq \mathbb{R}^n$ denote the set of all maximal and minimal velocity bounds of the DOF. The allowed DOF velocities are defined by the abilities of the actuators and given by $\mathcal{V}_{al} \subseteq \mathcal{V}$.
4. Let $L_s : \mathcal{Q} \to \mathcal{S}$, where $\mathcal{S} \subseteq \mathbb{R}^e$ and $e \leq n$, be a map from the configuration space to an e-dimensional sub-space in $\mathcal{Q}$. We constrain a trajectory $\phi(t) \in \mathcal{Q}$ to be on the sub-space such that

$$L_s(\phi(t)) \in \mathcal{S} \subset \mathcal{Q} \tag{2}$$

for all $t \in [0, t_g]$. This is a formulation of an equality constraint and examples for such could be: maintaining a coffee cup upright or sliding a manipulator's end-effector on a surface. Limitations on $L_s(\cdot)$ will be set further on.

A trajectory $\mathbf{Œ}(t) \in \mathcal{Q}$ is said to be a *feasible trajectory* in $t \in [0, t_g]$ if it satisfies the above constraints at that time frame.

3.3 Optimality criterion

Let $H : \mathcal{Q} \times \mathcal{U} \times \Upsilon \to \mathbb{R}_{\geq 0}$ be a scalar cost functional that maps a trajectory to a non-negative cost. An optimal trajectory is the one that minimizes the given cost-functional $H(\phi(t), \mathbf{u}(t), t_g)$ where t_g is the time when the system reaches the goal state. Examples of possible cost functions to minimize could reflect: the operation time, actuator's torques/forces, energy consumption, the negative of the proximity from the boundaries of the kinodynamic constraints, etc.

3.4 Motion planning problem

The motion planning problem under kinodynamic constraints is defined as follows. Given the desired goal state $\mathbf{q}(t_g) = [\phi(t_g)\ \dot{\phi}(t_g)]^T$ of the system and the cost function $H(\phi, \mathbf{u}, t_g)$, compute the optimal trajectory $\phi^*(t) \in \mathcal{Q}_{free}$, $\dot{\phi}^*(t) \in \mathcal{V}_{al}$ for the dynamic system to reach the goal state at some finite time $t_g \in [0, T]$ where $T \in \Upsilon$. The open-loop control input $\mathbf{u}^*(t) \in \mathcal{U}_{al}$ for $t \in [0, t_g]$ would also be provided to form an optimal trajectory. The required initial state of the motion would be the result of $\phi^*(0)$.

3.5 Assumptions

The assumptions for this work are as follows:

1. We assume full knowledge of the system's dynamics.
2. The trajectories of the dynamic obstacles are fully known.
3. The goal position is in $\mathcal{Q}_{free}$ and its desired velocity is in $\mathcal{V}_{al}$.
4. For practical applications, as stated in the introduction, the trajectory's initial velocity is considered zero. This assumption can be easily released as will be discussed in Sect. 4.2.

4 Preliminaries

In this section we present some notions that are used in the proposed algorithm.

4.1 Configuration and task spaces

This paper uses the notion of the *configuration space* (C-space) to parameterize the general coordinates of the dynamical system. Moreover, we use the notion of the *Task Space* (T-space) to parameterize some coordinates of the system that are essential to the designated work task.

Let $\mathcal{T} \subseteq \mathbb{R}^m$ be the T-space of the dynamic system given in Eq. (1), where m is dimension of the T-space, and let $\mathbf{p}(t) \in \mathcal{T}$ be some task of the system at time t. For example, a task of a robotic manipulator is the end-effector's position and orientation. Moreover, let $\mathcal{T}_{free}$ denote the allowed T-space corresponding to the allowed C-space $\mathcal{Q}_{free}$. Transformation from the C-space to T-space is performed using the direct kinematics of the system. Therefore, we define a transformation map $L_M : \mathbb{R}^n \to \mathbb{R}^m$ based on the direct kinematics of the system. That is,

$$\mathcal{T} = L_M \circ \mathcal{Q}. \tag{3}$$

An arbitrary configuration $\phi(t) \in \mathcal{Q}_{free}$ is mapped to a task $\mathbf{p}(t) \in \mathcal{T}_{free}$ according to

$$\mathbf{p}(t) = L_M(\phi(t)). \tag{4}$$

Mapping of a velocity in the C-space to velocity in the T-space is given by

$$\dot{\mathbf{p}}(t) = J\dot{\phi}(t), \tag{5}$$

where $J = J(\phi(t)) = \frac{d\mathbf{p}}{d\phi}$ is the Jacobian matrix of the system. Same could be done to the task acceleration

$$\ddot{\mathbf{p}}(t) = J\ddot{\phi}(t) + \dot{J}\dot{\phi}(t). \tag{6}$$

In this paper, the planning is performed in the T-space for generality and demonstration of using both C-space and T-space. However, without loss of generality, the planning could also be performed in the C-space with no reference to the T-space.

4.2 Trajectory parameterization

We propose a parameterization formulation for a trajectory to reach the goal state. First we define an optional candidate trajectory that could complete the task.

Definition 2 A trajectory function $\mathbf{s}(t) \in \mathcal{T}$ is a candidate trajectory if it is twice differentiable and satisfies some h boundary constraints.

That is, in our motion planning problem, a trajectory $\mathbf{s}(t)$ is a candidate trajectory if it satisfies the following $h = 3m$ boundary constraints that impose the initial velocity (according to Assumption 4) and final state:

$$\begin{cases} \dot{\mathbf{s}}(0) = \mathbf{0} \\ \mathbf{s}(t_g) = L_M(\phi(t_g)). \\ \dot{\mathbf{s}}(t_g) = J\dot{\phi}(t_g) \end{cases} \tag{7}$$

Note that, although out of the scope of this paper, if one desires to constrain an initial position as well, then m more boundary constraints are added for $\mathbf{s}(0)$. Similarly, if we wish to have a non-zero initial velocity (to remove Assumption 4), the first constraint in (7) should be removed and the velocity parameters must be added to the parameterization vector described next.

The following definition describes a candidate trajectory function that is constrained by the problems boundary constraints and has redundant parameters to optimize.

Definition 3 A candidate trajectory function $\mathbf{s}(t) = f_s(t, \mathbf{w}) \in \mathcal{T}$, where $\mathbf{w} = [w_1 \cdots w_{m\cdot k}]^T \in \mathbb{R}^{m\cdot k}$ and has h boundary constraints, is redundant if $m \cdot k > h$ for $k \in \mathbb{R}^+$.

The parameters in $\mathbf{w}$ are coefficients of the trajectory function $\mathbf{s}(t)$ that can be taken as, for example, coefficients of a polynomial function or of a Fourier series. The three constraints in (7) impose the values for w_i, $i = 1, \ldots, h$ and leave $d = m\cdot k - h$ free parameters for the function to be optimized. Moreover, the goal time t_g can also be chosen as a free parameter if no time constraint is imposed. Therefore, the redundant parameters are denoted as $\sigma = [w_{h+1} \cdots w_{m\cdot k}\ t_g]^T \in \Omega$, where $\Omega \subseteq \mathbb{R}^d \times \Upsilon$. If the goal time t_g is fixed and known, it should not be included in the parameters vector σ. The parameterization vector σ is therefore, the set of free parameters of $\mathbf{s}(t)$ that are not constrained by (7). As such,the boundary constraints are imposed on function $\mathbf{s}(t) = f_s(t, \sigma)$ while providing a desired number of free parameters in σ for motion planning and optimization. The following is an example of the parameterization.

Example 1 Consider a one-dimensional system with the following boundary constraints: $\dot{s}(0) = 0$, $s(t_g) = s_g$, and $\dot{s}(t_g) = v_g$. The trajectory can be chosen as an 7-dimensional polynomial function of the form:

$$s(t) = \sum_{n=0}^{7} a_n t^n. \tag{8}$$

Therefore, the constraints can be written as

$$\begin{cases} a_0 = 0 \\ \sum_{n=0}^{7} a_n t_g^n = s_g \\ \sum_{n=1}^{7} n a_n t_g^{n-1} = v_g \end{cases}. \tag{9}$$

Thus, a_0 and two more parameters, say a_1 and a_2, are defined in terms of the remaining parameters, $a_3, \ldots, a_7$ and the goal time t_g. Hence, the parameterization vector is defined by these six free parameters to be

$$\sigma = [a_3, a_4, a_5, a_6, a_7, t_g]^T. \tag{10}$$

$s(t) = f_s(t, \sigma)$ is now a function of the values in σ and the goal time. The trajectory in the configuration space can be expressed as

$$\phi(t) = L_M^{-1}(s(t)), \quad \dot{\phi}(t) = J^{-1}\dot{s}(t). \tag{11}$$

These expressions can now be used to reformulate the constraints as described in Sect. 5. □

4.3 Open-loop control input

Given a desired task trajectory $\mathbf{p}(t) = f_p(t, \sigma)$ defined by σ, an expression of the desired open-loop control force/torque is required. Using Eqs. (4)–(6), the expression for $\phi(t)$, $\dot{\phi}(t)$ and $\ddot{\phi}(t)$ in terms of σ are given by

$$\phi(t, \sigma) = L_M^{-1}\left(f_p(t, \sigma)\right), \tag{12}$$

$$\dot{\phi}(t, \sigma) = J^{-1}\left(\dot{f}_p(t, \sigma)\right) \tag{13}$$

and

$$\ddot{\phi}(t, \sigma) = J^{-1}\left(\ddot{f}_p(t, \sigma) - \dot{J}\dot{\phi}(t)\right), \tag{14}$$

where $J = J(\phi(t, \sigma)) = J(L_m^{-1} f_p(t, \sigma))$. We assume the existence of $L_m^{-1}(\cdot)$ and $J^{-1}(\cdot)$ due to the fully actuated system in (1). Substituting (12)–(14) in (1) yields an expression for $\mathbf{u}(t) \in \mathcal{U}$:

$$\mathbf{u}(t) = \Theta(t, \sigma) \text{ or } \Theta(t, \sigma) - \mathbf{u}(t) = 0, \tag{15}$$

where $\Theta(t, \sigma) = M\ddot{\phi}(t, \sigma) + C\dot{\phi}(t, \sigma) + G$. One may see $\Theta(t, \sigma) = 0$ as a time-varying d-dimensional hyper-surface in Ω expressing the dynamics of the system. To track a trajectory $\phi(t)$ parameterized by σ, the hyper-surface must contain σ for all times $t \in [0, t_g]$. Therefore, in expression (15) we have the open-loop control input $\mathbf{u}(t)$ which is used to force

the hyper-surface to contain σ throughout the motion. Hence, satisfying condition (15) is required to follow the desired trajectory. Another way to look at (15) is as an inverse dynamics (Spong et al. 2006) expression in terms of t and σ. We choose the open-loop control input, i.e., force/torque $\mathbf{u}(t)$, such that the constraint is satisfied.

We have acquired expressions for the task response which depend on the choice of the free parameter σ of the problem. Therefore, we seek to find a choice of σ that will provide an optimal trajectory under the kinodynamic constraints.

5 Constraints formulation

In this section we formulate the constraints of the system's motion in terms of the free parameters of the problem. We formulate C-space constraints that define $\mathcal{Q}_{free}$, other constraints in the T-space, and finally velocity and torque constraints imposed by the limitations of the system. In the following section, we denote the ith component of a vector with index i in the subscript, e.g., $(\cdot)_i$.

As mentioned in Sect. 3.2, it is possible that not all the C-space $\mathcal{Q}$ is accessible. Therefore, we can formulate the free space $\mathcal{Q}_{free}$ explicitly as a set of z_1 constraints $\Phi \in \mathbb{R}^{z_1}$

$$\Phi_i(\phi(t)) \leq 0, \ \forall i = 1, \ldots, z_1. \tag{16}$$

The set of constraints is composed of limitations of the actuator's range and forbidden regions due to obstacles. The set of constraints in Eq. (16) could also be written in terms of the task using (12) as

$$\Phi_i\left(L_M^{-1}(f_p(t,\sigma))\right) \leq 0, \ \forall i = 1, \ldots, z_1. \tag{17}$$

Note that if some obstacles cannot be described explicitly, conventional sampling-based collision checking can also be applied in the algorithm straight forward. Other constraints in the T-space could be formulated as $g_c \in \mathbb{R}^{z_2}$ by

$$g_{c_i}(\mathbf{p}(t)) = g_{c_i}\left(f_p(t,\sigma)\right) \leq 0, \ \forall i = 1, \ldots, z_2. \tag{18}$$

The set of allowed velocities $\mathcal{V}_{al}$ in C-space is the actuator's velocities within defined bounds for each joint i

$$\omega_{min_i} \leq \dot{\phi}_i(t) \leq \omega_{max_i}, \ \forall i = 1, \ldots, n. \tag{19}$$

Using (13), the velocity constraints of (19) in terms of t and σ are equivalent to

$$\begin{cases} \left(J^{-1}\frac{\partial f_p(t,\sigma)}{\partial t}\right)_i - \omega_{max_i} \leq 0 \\ -\left(J^{-1}\frac{\partial f_p(t,\sigma)}{\partial t}\right)_i + \omega_{min_i} \leq 0 \end{cases}, \ \forall i = 1, \ldots, n, \tag{20}$$

if both are satisfied.

The same could be done to constrain the necessary forces/torques such that $\mathbf{u}(t) \in \mathcal{U}_{al}$. Explicitly we can set upper and lower bounds to the open-loop control input required to track the trajectory such that

$$u_{min_i} \leq u_i(t) \leq u_{max_i}, \ \forall i = 1, \ldots, n. \tag{21}$$

Therefore, in terms of t and σ and according to (15), the force/torque constraints are written as

$$\begin{cases} \Theta_i(t,\sigma) - u_{max_i} \leq 0 \\ -\Theta_i(t,\sigma) + u_{min_i} \leq 0 \end{cases}, \ \forall i = 1, \ldots, n \tag{22}$$

and are equivalent to (21) if both are satisfied. The set of inequalities given in (17), (18), (20), (22) could be written as

$$\Psi_i(t,\sigma) \leq 0, \ i = 1, \ldots, z, \tag{23}$$

where $\Psi(t,\sigma)$ is a set of z functions:

$$\Psi(t,\sigma) = \begin{pmatrix} \Phi\left(L_M^{-1}(f_p(t,\sigma))\right) \\ g_c\left(f_p(t,\sigma)\right) \\ J^{-1}\frac{\partial f_p(t,\sigma)}{\partial t} - \omega_{max} \\ -J^{-1}\frac{\partial f_p(t,\sigma)}{\partial t} + \omega_{min} \\ \Theta(t,\sigma) - \mathbf{u}_{max} \\ -\Theta(t,\sigma) + \mathbf{u}_{min} \end{pmatrix}_{z\times 1} \tag{24}$$

and $z = z_1 + z_2 + 4n$.

Equation (23) expresses the inequality constraints of the problem. Further, we must formulate the equality constraints presented in (2) in terms of σ. Explicitly, a trajectory $\phi(t)$ is constrained to be on the sub-space $\mathcal{S}$ for all times $t \in [0, t_g]$, if it satisfies an equality constraint of the form $\mathbf{h}_e(\phi(t)) = 0$. Using (12) we can acquire the constraint

$$\mathbf{h}_e(t,\sigma) = 0. \tag{25}$$

Equality constraint (25) is a time-varying surface in Ω. However, the aim is to find a single σ which is on the surface at all times. Therefore, we impose limitations on function $\mathbf{h}_e(t,\sigma)$ such that $\mathbf{h}_e \in C^l$ is continuously differentiable l times, where $l < \infty$ and there exist a time-invariant region $\bar{\mathbf{h}}_e$ in $\mathbf{h}_e$ obtained by explicitly solving the set of equations

$$\mathbf{h}_e^{(j)}(t,\sigma) = 0 \tag{26}$$

where $\mathbf{h}_e^{(j)}$ is the jth time derivative of $\mathbf{h}_e$ for $j = 1, \ldots, l$. Therefore, the new time-invariant equality constraint will be

$$\bar{\mathbf{h}}_e(\sigma) = 0. \tag{27}$$

We define $\mathcal{H} \subset \Omega$ as the subset containing the points which satisfy (27). If no equality constraints exist, then $\mathcal{H} = \Omega$. If formulation (27) could not be obtained by solving (26), it means that a time-invariant region does not exist in h. Therefore, a single σ satisfying (25) could not be obtained and there is no solution to the desired trajectory optimization problem.

The inequalities in (23) define the feasible region of the dynamic system in terms of time and the desired trajectory parameter σ. That is, we obtained a set of analytical constraints that defines a time-varying region in Ω. The next section presents the time-varying constraint problem and the search algorithm to find an optimal trajectory satisfying the constraints.

6 Feasibility search algorithm

In this section we define the trajectory feasibility problem and present an algorithm for solving it.

6.1 Time-varying constraint (TVC) problem

In the previous section we obtained a set of inequalities depending on t and the parameters vector σ. Recall that the components in σ are independent of the time. Therefore, we would like to find an optimal vector $\sigma^* \in \Omega$ that satisfies the constraints from initial to goal time and minimizes some cost function. Such an optimal vector will sufficiently define the motion of the system under the kinodynamic constraints. Let $\Sigma \subset \Omega$ be a user defined allowed region for σ. The range of set Σ is chosen in a pre-processing step where the user simulates sets of points in Ω with different ranges. The user then chooses the range that best fits the workspace of the robot. The range for the time parameter in Σ is chosen according to the allowed time frame for the motion. Next, we define the notion of a feasible set.

Definition 4 A set $\Omega_f \subset \Omega$ is a feasible set in $t \in [t_1, t_2]$ if $\Omega_f \subseteq \Sigma$ and each $\sigma \in \Omega_f$ satisfies inequality (23) and equality (27) for all time $t \in [t_1, t_2]$.

We now define a feasible vector.

Definition 5 A vector of trajectory parameters $\sigma \in \Omega$ is said to be feasible in $t \in [t_1, t_2]$ if $\sigma \in \Omega_f$.

The above two definitions conclude that a vector is feasible if

$$\sigma = \{\sigma \in \Omega_f | \sigma \in \Sigma, \Psi_i(t, \sigma) \leq 0, \bar{\mathbf{h}}_e(\sigma) = 0, \\ \forall t \in [t_1(\sigma), t_2(\sigma)]\} \tag{28}$$

for all $i = 1, \ldots, z$. Notice that the time interval is written in general $[t_1(\sigma), t_2(\sigma)]$ and is a function of σ. This is due to the definition of the free parameters vector σ, which could include parameters that define the operation time. Therefore, the choice of σ determines the boundary time. We now face the problem of finding the feasibility set $\Omega_f \subseteq \Sigma$ where for all vectors within it, inequality (23) is maintained at all times. Formally, the problem is as follows.

Problem 1 Given the set of constraints in (23) and the set Σ, find the feasibility set $\Omega_f \subseteq \Sigma$.

Solving the above problem provides the feasible set Ω_f from which the optimal solution is to be chosen. Hence, we define the following minimization problem.

Problem 2 Find the vector $\sigma^* \in \Omega_f$ where $\Omega_f \subseteq \Sigma$ such that

$$\begin{aligned} \sigma^* = \arg\min_{\sigma} \quad & H(\sigma) \\ \text{subject to} \quad & \sigma^* \in \Omega_f \end{aligned} \tag{29}$$

where $H(\sigma)$ is some cost function to minimize.

In other words, the above general problem is finding an optimal vector σ^* that is feasible and minimizes some cost function $H(\sigma)$. Figure 1 illustrates the two problems. The position and volume of $\Psi(t, \sigma)$ in Ω varies in time and therefore, the solution of the problem is in a domain formed by projecting the constraints for time t_1 to t_2 on space Ω. The intersection formed by these projections, if one exists, is the feasibility domain that the optimal solution σ^* should be chosen from.

The above problem seeks a feasibility set over a period of time. It is important to note that we seek a feasibility set if such exists. In the next subsection we present an algorithm for finding the feasibility set and choosing an optimal solution

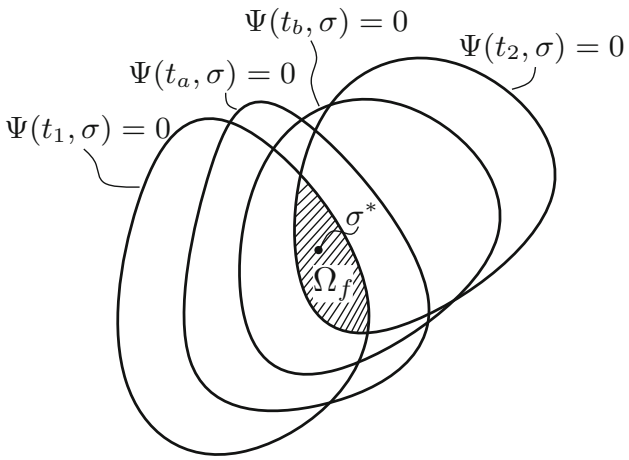


Fig. 1 The TVC problem where $t_1 < t_a < t_b < t_2$

from it. The probability to find a solution if one exists is also presented.

6.2 Feasibility search algorithm

A search algorithm is now presented to find the set Ω_f of feasible vectors. The domain formed by the set of constraints in inequality (23) is non-linear, non-convex, and not continuous. Therefore, an analytical solution of the reachable set is only possible in rare and simple instances. We present a numerical search algorithm to find a set of vectors satisfying the above constraints. Further, we can choose one vector from the set that best minimizes the cost function. We begin by presenting a simple definition for normalizing the time interval.

Lemma 1 *A vector σ is feasible in $t \in [t_1, t_2]$ if the constraint $\Psi_i(\lambda t_1 + (1-\lambda)t_2, \sigma) \leq 0$ is satisfied for all $0 \leq \lambda \leq 1$ and for all $i = 1, \ldots, z$.*

Proof For each time instant $t \in [t_1, t_2]$ there exists λ such that $t = \lambda t_1 + (1-\lambda)t_2$ and $0 \leq \lambda \leq 1$. Therefore, if a vector σ satisfies $\Psi_i(t, \sigma) \leq 0$ for all $i = 1, \ldots, z$ and $t_1 \leq t \leq t_2$, it must also satisfy $\Psi_i(\lambda t_1 + (1-\lambda)t_2, \sigma) \leq 0$ for all $i = 1, \ldots, z$ and $0 \leq \lambda \leq 1$. □

Lemma 1 is utilized as a criterion for determining whether a vector σ is feasible. Numerically, for σ we check the constraint for time $t = \lambda t_1 + (1-\lambda)t_2$ with $\lambda = \{0, \Delta\lambda_1, \Delta\lambda_1 + \Delta\lambda_2, \ldots, 1\}$. The value of the step $\Delta\lambda_j$ will be further defined. Figure 2 illustrates an abstraction of the feasibility problem and the line in time defined by Lemma 1. Without loss of generality, from this point we will address the problem with the time frame $[t_1, t_2] = [0, t_g]$.

The basis of the algorithm's operation is selecting a set of N random points within Σ and checking each for its feasibility. The feasibility search algorithm is presented as the `Feasibility_Search(·)` function in Algorithm 1. The

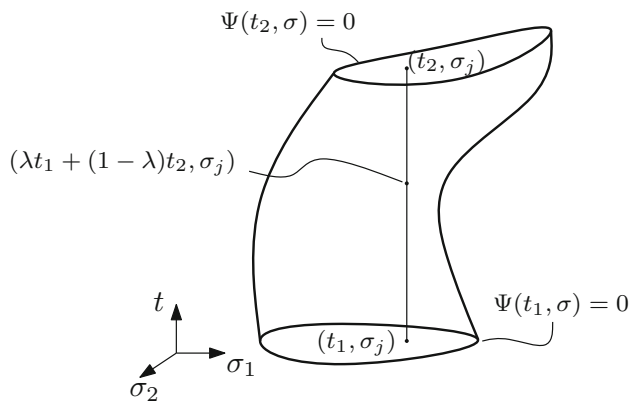


Fig. 2 Vector σ satisfying Lemma 1

algorithm's input is the allowed set Σ chosen the user, equality set $\mathcal{H}$ and the set of constraints of Eq. (23). The first step of the algorithm is to determine the number of random points N such that the probability to find a solution is more than a user defined probability $1 - P_{max}$. The calculation of N based on the choice of P_{max} will be presented later in the algorithm's analysis. The next step is to sample N random points $\mathcal{P} = \{\sigma_1, \ldots, \sigma_N\}$ uniformly distributed in Σ. The allowed region formed by Σ is a hyper-rectangle in Ω and therefore we sample points in each axis of Ω within the boundaries defined by Σ. Such sampling provides a Poisson distribution over the volume of Σ. However, due to the equality constraint (27), we sample points in $\Sigma \cap \mathcal{H}$. That is, we sample points that are in Σ and satisfy (27). Thus, imposing the equality constraint on the desired trajectory.

The next step is going over all the N points in $\mathcal{P}$ and filtering out those that are not feasible. The final time $t_{g_i}(\sigma_i)$ is determined for each point σ_i checked. We check if the constraints are satisfied for time $t = \lambda t_{g_i}$ where $\lambda = \{0, \Delta\lambda_1, \Delta\lambda_1 + \Delta\lambda_2, \ldots, 1\}$. Those that do not satisfy the constraints are eliminated and the filtered set $\mathcal{P}$ with size $M \leq N$ is outputted.

Algorithm 1 `Feasibility_search`$(\Sigma, \mathcal{H}, \Psi, P_{max}, \epsilon_b)$

Input: The allowed set Σ, equality set $\mathcal{H}$, set of constraints Ψ, the probability P_{max}, and tolerance ϵ_b.
Output: Set of feasible points Ω_f.

```
Calculate number of random points N such that the probability to find a solution is more than 1 − P_max.
Generate the set P = {σ_1, ..., σ_N} of N uniformly distributed random points within Σ ∩ H.
Calculate S_max. // using optimization prob. in (32).
for δ_S = ΔS --ΔS--> 1 do
    for i = 1 → N do
        if ¬(Adaptive_Check(σ_i, Ψ, δ_S · S_max, ε_b)) then
            Remove σ_i from P.
        end if
    end for
end for
return Ω_f = P = {σ_1, ..., σ_M}    // M ≤ N
```

Scanning the constraint $\Psi(\lambda t_{g_i}, \sigma_i)$ for $\lambda = \{0, \Delta\lambda, 2\Delta\lambda, \ldots, 1\}$ where $\Delta\lambda$ is a constant value is rather risky. The value of Ψ might ascend over 0 and descend below again within the discretized step size. An example of such is shown in Fig. 3 where with step size above 0.03 failure of the constraints might not be discovered. Moreover, too small step sizes could be unnecessary and the price would include very high complexity. Therefore, we present a simple adaptive step size algorithm to fine tune the time steps and diagnose or rule out such scenarios.

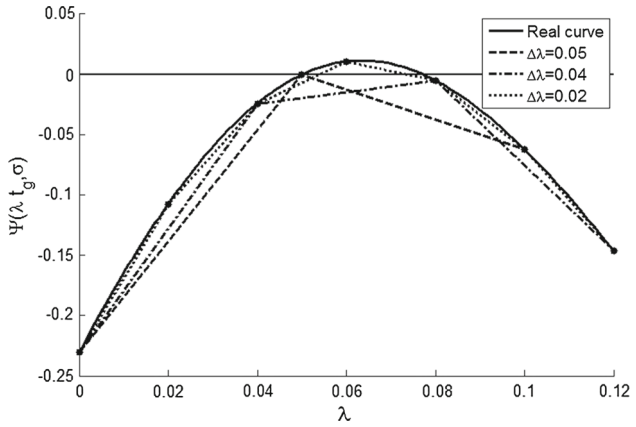


Fig. 3 The maximum constraint value along time with change of the step size $\Delta\lambda$

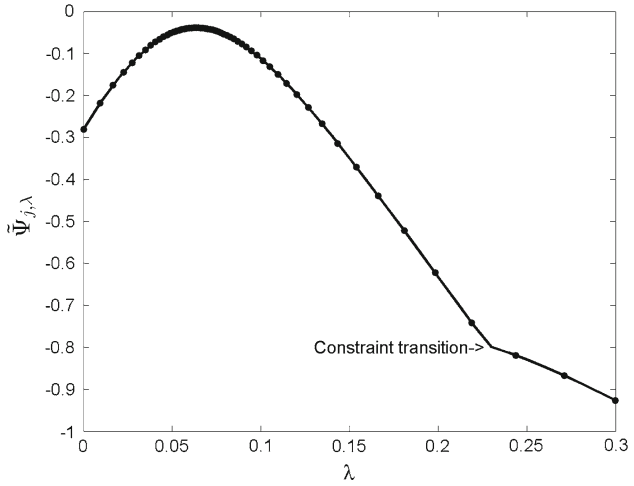


Fig. 4 Adaptive step size algorithm

Definition 6 The constraint value of a feasible point σ_i at time λt_{g_i} is defined to be

$$\tilde{\Psi}_{i,\lambda} = \max_j \left\{ \Psi_j(\lambda t_{g_i}, \sigma_i) \right\}, \tag{30}$$

where Ψ_j is the jth component of the constraint vector Ψ.

That is, the constraint value is the shortest distance at time λt_{g_i} from point σ_i to the boundary of the closest constraint. Notice that we refer to a distance with a positive value, but the value of $\tilde{\Psi}_{i,\lambda}$ is maintained negative for an indication that σ_i is a feasible point. Assume that the change rate of the constraint value with regards to λ is bounded by

$$\frac{\Delta\tilde{\Psi}_{i,\lambda}}{\Delta\lambda} \leq S_{max}, \ \forall\, 0 \leq \lambda \leq 1. \tag{31}$$

That is, the maximum slope of the constraint value $\tilde{\Psi}_{i,\lambda}$ is S_{max}. Under this assumption we can say that if at time λt_{g_i} the constraints are satisfied, $\tilde{\Psi}_{i,\lambda} < 0$, then the minimum time for the constraint to reach 0 is $(\lambda + \Delta\lambda_{min})t_{g_i}$ where $\Delta\lambda_{min} = -\frac{\tilde{\Psi}_{i,\lambda}}{S_{max}}$. Therefore, as we get closer to a boundary of a constraint, we decrease $\Delta\lambda$ such that reaching above the zero line in that time frame is not possible. Figure 4 illustrates the selection of $\Delta\lambda$ as it gets smaller when approaching the zero line and larger when receding. However, in this adaptive approach, even though $\tilde{\Psi}_{i,\lambda}$ passes the zero line, the algorithm will never do so as it will continue to decrease $\Delta\lambda$. Therefore, we bound such that the algorithm will stop checking the current σ_i (and remove it) if $\tilde{\Psi}_{i,\lambda} < \epsilon_b < 0$, where ϵ_b is a value that will be defined further in the algorithm's analysis. This also serves as a safety distance, assuring the solution is far enough from the constraints boundary. To calculate S_{max} we differentiate the constraint vector by λ to acquire its slope $\frac{\partial\Psi(\lambda t_g,\sigma)}{\partial\lambda}$, where t_g is the maximum possible goal time based on the allowed time interval given in Σ. S_{max} is the maximum slope of all components over all time and can be computed by the following maximization problem

$$\begin{aligned} S_{max} = \max_{\lambda,\sigma,j} \quad & S_j(\lambda t_g, \sigma) \\ \text{subject to} \quad & 0 \leq \lambda \leq 1 \\ & \sigma \in \Sigma \end{aligned} \tag{32}$$

where S_j is the jth component of the constraints derivative $S(\lambda t_g, \sigma) = \frac{\partial\Psi(\lambda t_g,\sigma)}{\partial\lambda}$. This could be computed analytically using Kuhn–Tucker conditions (Kuhn and Tucker 1950) or numerically. The adaptive step size function $Adaptive\_Check(\cdot)$ is presented in Algorithm 2.

Algorithm 2 `Adaptive_Check`$(\sigma_i, \Psi, S_{max}, \epsilon_b)$

Input: σ_i, the set of constraints Ψ, slope S_{max} and the tolerance ϵ_b.
Output: Boolean: 1 if σ_i is feasible, 0 if not feasible.

```
Set λ = 0.
Extract t_{g_i} from the last component of σ_i.
while ( λ ≤ 1 ) do
    Calculate Ψ̃_{i,λ} = max_j {Ψ_j(λ t_{g_i}, σ_i)}.
    if ¬( Ψ̃_{i,λ} < ε_b ) then
        Return 0.
    else
        Calculate Δλ = −Ψ̃_{i,λ} / S_max.
        λ = λ + Δλ.
    end if
end while
Return 1.
```

The adaptive step size with the maximum slope S_{max} provides efficient sweep of the generated points by reducing the number of checks for each point in $\mathcal{P}$. However, for high-dimensional systems such adaptive search by its own can take a long time. Therefore, we propose a filtering process in which we start with a fraction $0 \leq \delta_S \leq 1$ of S_{max}. That

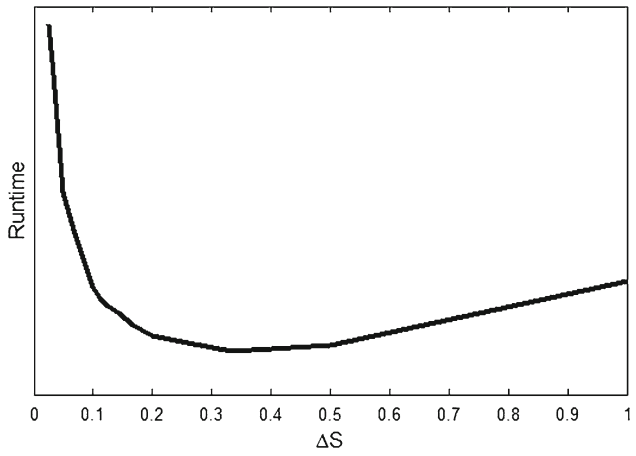


Fig. 5 Typical runtime with respect to change of ΔS

is, in the first iteration of the `Adaptive_Check`$(\cdot)$ function, the maximum allowed slope will be $\delta_S \cdot S_{max}$. Vectors which are diagnosed to be non-feasible with maximum slope of $\delta_S \cdot S_{max}$ will surely be non-feasible with the full maximum slope S_{max}. Therefore we can eliminate them with less checks. In the next iterations δ_S is increased and so on, until the last iteration where $\delta_S = 1$. In this process, with small δ_S in the beginning, large time steps would be taken to reduce runtime avoiding excessive calculations. For each iteration we take the fraction as $\delta_S = i \cdot \Delta S$ where ΔS is the step size, $i = 1, \ldots, b$ and $b\Delta S = 1$. Hence, the following problem is the determination of the step size ΔS. Small ΔS can cause excessive iterations which could cause the opposite and increase the runtime. Experiments on the algorithms runtime on several systems have revealed a typical behavior as shown in Fig. 5. The optimal step size should be $\Delta S = \frac{1}{3}$ which decreases the runtime by an average of 50%.

In high-dimensional systems, computation of the dynamic system (Eq. (15)) takes a large percentage of the runtime. Therefore, in addition to filtering-out non-feasible solutions using δ_S, we can separate the torque constraint check from the other constraints. In that way, only candidate trajectories which passed the kinematic constraints will be checked for the torque constrains. Experiments have shown a 90% runtime reduction compared to a non-separated constraint check.

6.3 Optimal solution

The final step of the algorithm is selecting the optimal solution among the set of feasible points $\Omega_f = \{\sigma_1, \ldots, \sigma_M\}$ found in Algorithm 1 and perform local fine-tuning optimization. Given the cost function $H(\sigma)$, the optimal solution σ^* is found according to Algorithm 3. In this algorithm, first a simple naive search is performed on the feasibility set Ω_f to find a feasible point σ_k that best minimizes $H(\sigma)$ (Line 1). Recall

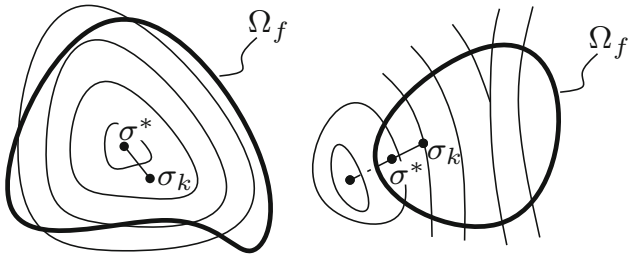


Fig. 6 Two examples of the optimal solution refinement; one (left) descended to the local minimum while the other (right) was stopped by the constraint boundary

that the trajectory is an analytical function and therefore, in most cases, the derivative of $H(\sigma)$ can be acquired. Thus, we can utilize a Gradient Descent (GD) method (Cauchy 1847) to refine the solution and find a local minimum in the neighborhood of σ_k. The GD method is an iterative algorithm with an update law (Line 5) of the form

$$\sigma^{(i+1)} = \sigma^{(i)} - \gamma \nabla H(\sigma^{(i)}) \tag{33}$$

where $\gamma > 0$ is chosen using exact or backtracking line search (Boyd and Vandenberghe 2004). In each iteration the constraints is checked. Notice that the `Adaptive_Check` function prevents the solution from approaching the constraint boundary with distance less than ϵ_b. The iterations are terminated if the new point breaks the constraints or if the convergence condition is satisfied (Line 10 of Algorithm 3). Figure 6 presents two examples of local refinement of the optimal solution; one was stopped by the constraint while the other managed to reach the local minimum. The convergence condition checks if the norm of the current descent is smaller than a predefined tolerance ϵ_d, that is, we have reached a local minimum with ϵ_d accuracy. The last point of the iteration is the optimal solution and is returned by the algorithm.

Algorithm 3 `Local_Optimization`(Ω_f, ϵ_d)

Input: Feasible set $\Omega_f = \{\sigma_1, \ldots, \sigma_M\}$ and tolerance ϵ_d.
Output: Optimal solution σ^*.
1: $k = \arg\min_i H(\sigma_i),\ i = 1, \ldots, M$
2: Define $\sigma^{(0)} = \sigma_k$.
3: $i = 0$.
4: **repeat**
5: $\quad \sigma^{(i+1)} = \sigma^{(i)} - \gamma \nabla H(\sigma^{(i)})$.
6: $\quad$ **if** $\neg$(`Adaptive_Check`$(\sigma^{(i+1)})$) **then**
7: $\quad\quad$ Return $\sigma^{(i)}$.
8: $\quad$ **end if**
9: $\quad i = i + 1$.
10: **until** $\left\|\nabla H(\sigma^{(i)})\right\|_2 < \epsilon_d$
11: $\sigma^* = \sigma^{(i)}$.
12: Return σ^*.

6.4 Overall algorithm

The full SKIP algorithm is presented in Algorithm 4. The algorithm receives input of the system's constraints in the form of inequalities (23), the desired probability P_{max} (will be presented in the next section), and the convergence tolerances ϵ_b and ϵ_d. If Algorithm 1 returns an empty set, the algorithm reports a failed search. Else, its final output is, according to (12), the desired optimal trajectory

$$\phi^*(t) = L_M^{-1}(\mathbf{f_p}(\mathbf{t}, \boldsymbol{\alpha}^*)), \tag{34}$$

The optimal open-loop control signals could also be obtained according to (15) and are

$$\mathbf{u}^*(t) = \Theta(t, \sigma^*). \tag{35}$$

Once a trajectory is found, trajectory tracking methods such as inverse-dynamics and PD control (Spong et al. 2006) can be applied straightforward.

7 Analysis

In this section a statistical and complexity analysis is performed for the proposed SKIP algorithm.

Algorithm 4 SKIP algorithm

Input: Set of constraints Ψ, probability P_{max}, and tolerances ϵ_b, ϵ_d.
Output: Optimal C-space trajectory $\phi^*(t)$.
1: Define allowed subset $\Sigma \in \Omega$
2: $\Omega_f = \texttt{Feasibility\_search}(\Sigma, \Psi, P_{max}, \epsilon_b)$.
3: **if** $\Omega_f = \emptyset$ **then**
4: **return** fail.
5: **end if**
6: $\sigma^* = \texttt{Local\_Optimization}(\Omega_f, \epsilon_d)$.
7: Output optimal trajectory $\phi^*(t) = L_M^{-1}(\mathbf{f_p}(\mathbf{t}, \boldsymbol{\alpha}^*))$

7.1 Statistical analysis

A statistical analysis is performed in this subsection to calculate the probability to find a solution if it exists. First, we define an operator that measures the volume of a set.

Definition 7 Let a map $\mathcal{L}_V : \Gamma \to \mathbb{R}_{>0}$ such that $\Gamma \subset \mathbb{R}^d$. The value of $\mathcal{L}_V \circ \Gamma$ is the hyper-volume measure of Γ according to the Lebesgue measure (Lebesgue and May 1966).

Let us assume Ω_f is known and let $0 \leq \rho \leq 1$ be the relative portion of Σ that is feasible; that is, the ratio between the hyper-volumes of Ω_f and Σ:

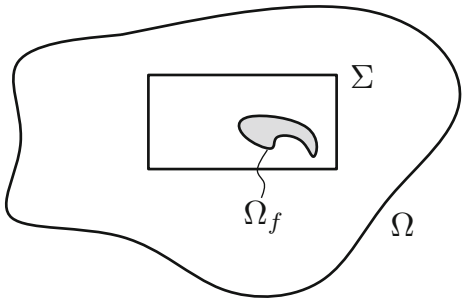


Fig. 7 Two-dimensional example of Σ and Ω_f in Ω

$$\rho = \frac{\mathcal{L}_V \circ \Omega_f}{\mathcal{L}_V \circ \Sigma}. \tag{36}$$

For simplification and practical usage, we will refer to Σ as a hyper-rectangle in Ω (Fig. 7). Let X_j be a random variable that takes the value 1 if $\sigma_j \in \Sigma$ falls within Ω_f and 0 if not. Thus, X_j is defined

$$X_j = \begin{cases} 1, & \sigma_j \in \Omega_f \\ 0, & \sigma_j \notin \Omega_f \end{cases}. \tag{37}$$

Because X_j can take only two values $\{0, 1\}$ and the probability of a random $\sigma_j \in \Sigma$ to fall in Ω_f is ρ, then $X_j \sim Bernoulli(\rho)$. Therefore, the probability for a random point σ_j to have a compatible random variable $X_j = y$ is

$$P(X_j = y) = \rho^y (1-\rho)^{1-y} \; for \; y \in \{0, 1\}. \tag{38}$$

If a set of N points $\mathcal{P} = \{\sigma_1, \ldots, \sigma_N\} \in \Sigma$ are independent, identically distributed random variables, and their compatible random variables X_i, $i = 1, \ldots, N$ are all Bernoulli distributed with success probability ρ, then the sum of the random variables is

$$Y = \sum X_i \sim Binomial(N, \rho). \tag{39}$$

Therefore, the probability to have k points in Ω_f for N random points ($k = 0, 1, \ldots, N$) is

$$P(Y = k) = \binom{N}{k} \rho^k (1-\rho)^{N-k} \tag{40}$$

where $\binom{N}{k}$ is the binomial coefficient representing the number of combinations distributing k successes in N random points. The Poisson distribution can be used as an approximation of the Binomial distribution if the number of random points N goes to infinity ($N \to \infty$) and the Bernoulli probability is sufficiently small ($\rho \to 0$). Therefore, $Y \sim Poisson(\bar{\lambda})$ where $\bar{\lambda} = N\rho$. That is,

$$P(Y=k) \simeq \frac{\bar{\lambda}^k}{k!} e^{-\bar{\lambda}}. \tag{41}$$

We determine that if a solution exists, the highest allowed probability that the solution will not be found ($k=0$) out of N random points is defined to be

$$P(Y=0|\rho) = e^{-N\rho}. \tag{42}$$

In the next Theorem we show the probability to find a solution in Σ if one exists.

Theorem 1 *The probability to find a solution in Σ, if one exists, approaches 1 as the number of generated random points approaches infinity.*

Proof The probability not to find a solution in Σ is given in (42). Therefore, the probability to find a solution is

$$P_s(N) = P(Y>0|\rho) = 1 - e^{-N\rho}. \tag{43}$$

If we increase N to infinity, then $e^{-N\rho}$ and the limit for the probability is

$$\lim_{N\to\infty} P_s(N) = 1. \tag{44}$$

That is, as N approaches infinity, the probability to find a solution in Σ approaches 1. □

However, taking infinite number of points is not feasible. Hence, we provide a formula for finding the number of random points to be taken given the highest allowed probability P_{max} not to find a solution. Based on (42), the number N of random points in Σ must be chosen such that

$$e^{-N\rho} \leq P_{max}. \tag{45}$$

This condition states that given ρ, the number of random points to be chosen such that if a solution exists, the probability not to find it is lower than P_{max}. That is, the probability to find a solution is higher than $1 - P_{max}$.

The last issue that must be addressed is how to determine ρ such that (45) could be used to calculate N given P_{max}. The determination of ρ is a resolution of how close we allow the desired parameters vector to be from the constraints boundaries. That is, how small do we allow the feasibility set to be in order to be considered to exist. If the volume of the feasibility set Ω_f is very small, the points within it must be very close to the constraints boundary. This is an undesired situation. Such a case must be bounded so that if Ω_f is too small, we would determine that a solution does not exist. For that matter, we propose a heuristic approach of approximating a characteristic measure of the constraints. We define a characteristic measure that is the average minimum distance from the center of the feasible region to the constraints boundaries over time. The centroid of the subspace formed by the constraints in (23) at time $\lambda_i t_g$ is calculated by

$$\sigma_{b_i} = \underset{\sigma}{arg\,min} \left\{ \sum_{j=1}^{z} a_{j,i} \Psi_j(\lambda_i t_g, \sigma) \right\}, \tag{46}$$

Ψ_j is the jth component of the constraint vector Ψ, $\lambda_i = 0, \overline{\Delta\lambda}, .., \beta\overline{\Delta\lambda}$, where $\beta = 1/\overline{\Delta\lambda}$, and t_g is the last component of vector σ. Because each constraint in Ψ has a different gradient, which affects the distance-value ratio, it is multiplied by its relative weight $a_{j,i}$ given by

$$a_{j,i} = \frac{\sum_{k=1}^{z,k\neq j} \Psi_k(\lambda_i t_g, \sigma)}{\sum_{k=1}^{z} \Psi_k(\lambda_i t_g, \sigma)}. \tag{47}$$

Hence, the average distance over time (with step size $\overline{\Delta\lambda} t_g$) is given by

$$r = \frac{1}{\beta} \sum_{i=1}^{\beta} \left(\frac{1}{z} \sum_{j=1}^{z} \Psi_j(\lambda_i t_g, \sigma_{b_i}) \right) \tag{48}$$

and is a characteristic measure of the constraint's size. Heuristically, the average is computed in β time steps and increasing β will increase the accuracy. Then, the minimum allowed volume Ω_{min} is defined as

$$\mathcal{L}_V \circ \Omega_{min} = (\alpha r)^d, \tag{49}$$

where $0 < \alpha \leq 1$ is the allowed fraction of r and is user-defined. That is, we define a hyper-cube with edge length αr and compute its volume to be the minimal allowed one. Thus, ρ is chosen as $\rho_{min} = (\mathcal{L}_V \circ \Omega_{min})/(\mathcal{L}_V \circ \Sigma)$ and the minimal number of random points N for a given P_{max} can be calculated according to (45) by

$$N \geq -\frac{1}{\rho} \log(P_{max}). \tag{50}$$

Moreover, the value of ϵ_b defined in Sect. 6.2 to be the minimum allowed distance from the constraint boundary is chosen using the characteristic measure calculated in (48) to be $\epsilon_b = \alpha r$.

In the following Theorem we conclude the probability to find a solution if such exists.

Theorem 2 *If a solution for the time-varying problem exists in Σ and N satisfies (50), the probability for the algorithm to find a solution is at least $1 - P_{max}$.*

Proof The probability not to find a solution if one exists is given in (42). If the algorithm selects N random points in the

allowed set Σ such that (50) is satisfied with given probability P_{max} and a minimum allowed volume ratio ρ_{min}, then substituting it in (42) yields

$$P(Y = 0|\rho_{min}) = P_{max}, \tag{51}$$

which is the probability not to find a solution. Hence, the probability to find a solution is $1 - P_{max}$. □

The choice of N is done ensuring finding a solution if such exists above a known probability $1 - P_{max}$. It should be noted that the local optimization performed in Algorithm 3 does not affect the probability to find a solution because it is performed only if a solution is found.

The following corollary extends Theorem 2 to calculate the probability to find a solution given a maximal calculation runtime T_c and assuming the time for computing one point is known to be τ_c.

Corollary 1 *Given a desired calculation time T_c and the computer cycle time τ_c to calculate one point, the probability to find a solution if one exists is*

$$P_s = 1 - e^{-\frac{T_c}{\tau_c}\rho}. \tag{52}$$

Proof The maximal number of random points that can be generated is $N_c = \left\lfloor \frac{T_c}{\tau_c} \right\rfloor$. Therefore, given ρ and according to (42), the probability not to find a solution is

$$P(Y = 0|\rho) = e^{-N_c\rho} = e^{-\frac{T_c}{\tau_c}\rho}. \tag{53}$$

Therefore, the probability to find a solution is $1 - e^{-\frac{T_c}{\tau_c}\rho}$. □

It is important to note that this analysis addresses the probability of finding a solution if one exists. It does not, however, prove that the result is indeed optimal. As like other sampling-based optimal planners (Karaman and Frazzoli 2011), optimality will be achieved if the planner is allowed to run long enough.

7.2 Complexity

The complexity of the proposed algorithm is presented next.

Theorem 3 *The upper bound time complexity of the proposed algorithm is in the order of $O(\xi N)$.*

Proof We generate N random points in Σ and therefore, the N points are to be checked for feasibility. Given the maximum constraint slope S_{max} and the maximum allowed boundary constraint distance ϵ_b. In the worst case, the constraint value $\tilde{\Psi}_{i,\lambda}$ approximately equals ϵ_b for all $0 \leq \lambda \leq 1$. Moreover, in the worst case, during the δ_S iterations, no points will be filtered-out but in the last iteration. Therefore, each time step size will be $\Delta\lambda = \frac{\epsilon_b}{S_{max}}$ and the number of time steps each σ_i will be checked for satisfying the constraints is $\xi = b\frac{1}{\Delta\lambda} = b\frac{S_{max}}{\epsilon_b}$. Hereafter, N random points will be checked, in the worst case, ξ times. Thus, the upper bound time complexity of the algorithm is in the order of $O(\xi N)$. □

Table 1 Kinematic and dynamic properties of the manipulator used in the simulations

Mass (kg)		Length (m)		C.O.M (m)	
m_1	22.3	L_1	0.24	l_1	0.021
m_2	21.2	L_2	0.25	l_2	0.127
m_3	8.51	L_3	0.10	l_3	0.032
m_4	7.58	L_4	0.19	l_4	0.090
m_5	1.34	L_5	0.05	l_5	0.008
m_6	1.85	L_6	0.14	l_6	0.058
Inertia (kg m^2)					
I_{x_1}	0.063	I_{y_1}	0.106	I_{z_1}	0.010
I_{x_2}	0.120	I_{y_2}	0.077	I_{z_2}	0.120
I_{x_3}	0.029	I_{y_3}	0.034	I_{z_3}	0.015
I_{x_4}	0.010	I_{y_4}	0.092	I_{z_4}	0.085
I_{x_5}	6.0×10^{-4}	I_{y_5}	9.8×10^{-4}	I_{z_5}	7.9×10^{-4}
I_{x_6}	1.9×10^{-3}	I_{y_6}	9.1×10^{-3}	I_{z_6}	7.7×10^{-3}

8 Simulations

In this section we demonstrate the algorithm's operation. We present a trajectory planning and optimization for a 6R manipulator for hitting a disk in a desired goal state. Here, we demonstrate planning for a high-dimensional system with no initial state, i.e., the initial state is left for planning. The algorithm was implemented in Matlab[1] on an Intel-Core i7-2620M 2.7 GHz laptop computer with 8 GB of RAM. The simulation videos can be seen on-line in resource 1.

The manipulator's dynamics is given by (1), where its physical properties are given in Table 1; m_i, I_i, L_i, and l_i are the mass, moment of inertia, length, and center of mass position (relative to the links joint) of link i, respectively. The joints' angles are $\phi(t) = [\varphi_1\ \varphi_2\ \varphi_3\ \varphi_4\ \varphi_5\ \varphi_6]^T$ and the end-effectors task is $\mathbf{p}(t) = L_M(\phi(t)) = [p_x(t)\ p_y(t)\ p_z(t)\ \varphi(t)\ \theta(t)\ \psi(t)]^T$, where φ, θ, ψ are the roll, pitch, and yaw, respectively.

The aim of the planning is to slide a cylinder, held by the end-effector, on an horizontal table. The cylinder must avoid static and dynamic obstacles, and hit a disk at a required position and velocity:

[1] Matlab is a registered trademark of The Mathworks, Inc.

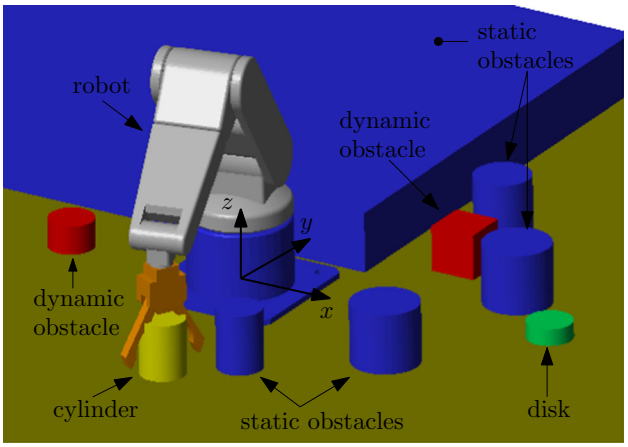


Fig. 8 The 6R simulation setup

$$\mathbf{p}(t_g) = \begin{pmatrix} -0.54\,[\text{m}] \\ 0\,[\text{m}] \\ 0\,[\text{m}] \\ 0\,[\text{rad}] \\ \pi/2\,[\text{rad}] \\ 0\,[\text{rad}] \end{pmatrix}, \quad \dot{\mathbf{p}}(t_g) = \begin{pmatrix} -1\,[\text{m/s}] \\ 0\,[\text{m/s}] \\ z_t\,[\text{m/s}] \\ 0\,[\text{rad/s}] \\ 0\,[\text{rad/s}] \\ 0\,[\text{rad/s}] \end{pmatrix}, \tag{54}$$

where z_t is the desired end-effector's height in order to maintain the grasped cylinder on the table. The setup for the simulation is seen in Fig. 8. The desired trajectory function $\mathbf{s}(t) \in \mathcal{T}$, where

$$\mathbf{s}(t) = (s_x(t)\ s_y(t)\ s_\theta(t))^T, \tag{55}$$

is given in the T-space by a g-order polynomial vector as

$$\mathbf{s}(t) = \sum_{i=0}^{g} \mathbf{m_i} t^i, \tag{56}$$

where $\mathbf{m_i} = [a_i\ b_i\ c_i\ d_i\ e_i\ f_i]^T$, $i = 0, \ldots, g$ are the coefficients of the polynoms. The desired goal state (54) and zero initial velocity impose the values for $\mathbf{m_1} = \dot{\mathbf{p}}(0) = \mathbf{0}$, $\mathbf{m_2}$, and $\mathbf{m_3}$. Hence, the parameters' vector is defined by the remaining coefficients and the goal time such that

$$\sigma = \left[\mathbf{m_0}^T\ \mathbf{m_4}^T \cdots \mathbf{m_g}^T\ t_g\right]^T, \tag{57}$$

and the desired trajectory is defined by σ such that $\mathbf{s}(t) = \mathbf{s}(t, \sigma)$. The static and dynamic obstacles are constraints in the form of

$$r_o^2 - (s_x(t) - x_b(t))^2 - (s_y(t) - y_b(t))^2 \leq 0, \tag{58}$$

where r_o is the sum of the cylinder and obstacle radii, $x_b(t)$ and $y_b(t)$ are obstacles' center coordinate. Note that for static obstacles, the center coordinates are constant in time. We use

Table 2 Angle, angular velocity and torque bounds of the manipulation used in the simulations

		Min. value	Max. value
Angles	$\phi_{1,4,5,6}$	$-180°$	$180°$
	ϕ_2	$-90°$	$30°$
	ϕ_3	$-150°$	$150°$
Angular velocity	$\omega_{1,4,6}$	-10 [RPM]	10 [RPM]
	$\omega_{2,3,5}$	-20 [RPM]	20 [RPM]
Torques	$u_{1,3}(t)$	-30 [Nm]	30 [Nm]
	$u_2(t)$	-120 [Nm]	120 [Nm]
	$u_{4,5,6}(t)$	-10 [Nm]	10 [Nm]

the desired trajectory (56) and the kinematic and dynamic model of the arm. By that, the obstacle constraints of (58), and the angle, angular velocity, and torque bounds shown in Table 2, are formed as (23) in terms of time and σ.

In addition to the inequality constraints above, we have some equality constraints. First, because the end-effector's motion is on a 2D plane parallel to the table, its height is constrained such that $s_z(t, \sigma) = z_t$. Moreover, we constrain the end-effector to be perpendicular to the table, that is, have constant pitch such that $\theta(t, \sigma) = \pi/2$. In more complex problems we would have to generate the random points on the constraints. However, these two equality constraints impose some values in σ, in particular, $c_0 = z_t$, $c_i = 0$, which are the coefficients for the $s_z(t)$ polynom, and $e_0 = \pi/2$ and $e_i = 0$, which are the coefficients of the pitch polynom $\theta(t)$, for $i = 1, \ldots, g$. These values could be maintained in σ or could be removed by reducing the dimension of σ, either way would not change runtime.

We have formed kinodynamic constraints as a function of σ and t in the form of Eqs. (23) and (24). Moreover, we have embedded the equality constraints in the structure of σ. Next, we set the parameters for the adaptive search. The desired trajectory polynomials $\mathbf{s}(t, \sigma)$ are chosen to be in the order of $g = 20$ and therefore the dimension of the parameter set is $\Omega \subseteq \mathbb{R}^{108} \times \Upsilon$. This value was chosen by means of trial and error to have enough free parameters for optimization but small enough not to negatively affect the performance. In the preprocessing step we chose Σ such that the free trajectory parameters are in the range of $a_i, b_i, d_i, f_i = [-10, 10] \times 10^{-12}$, $i = 0, 4, \ldots, g-1$, $a_g, b_g = [-0.7, 0.7]$, $d_g, f_g = [0, 2\pi]$ and the goal time interval to be $t_g = [0.5, 3.5]$. These values were set based on preliminary analysis showing the range of values for feasible polynomials. This step is currently manual and future work should consider ways for choosing the right values. Therefore, the set Σ is a 109-dimensional hyper-rectangle in Ω. Numeric calculation of ρ with user selection of $\alpha = 0.18$ (18% of the average distance) yields the relative portion $\rho = 0.0021$. After choosing the maximum probability

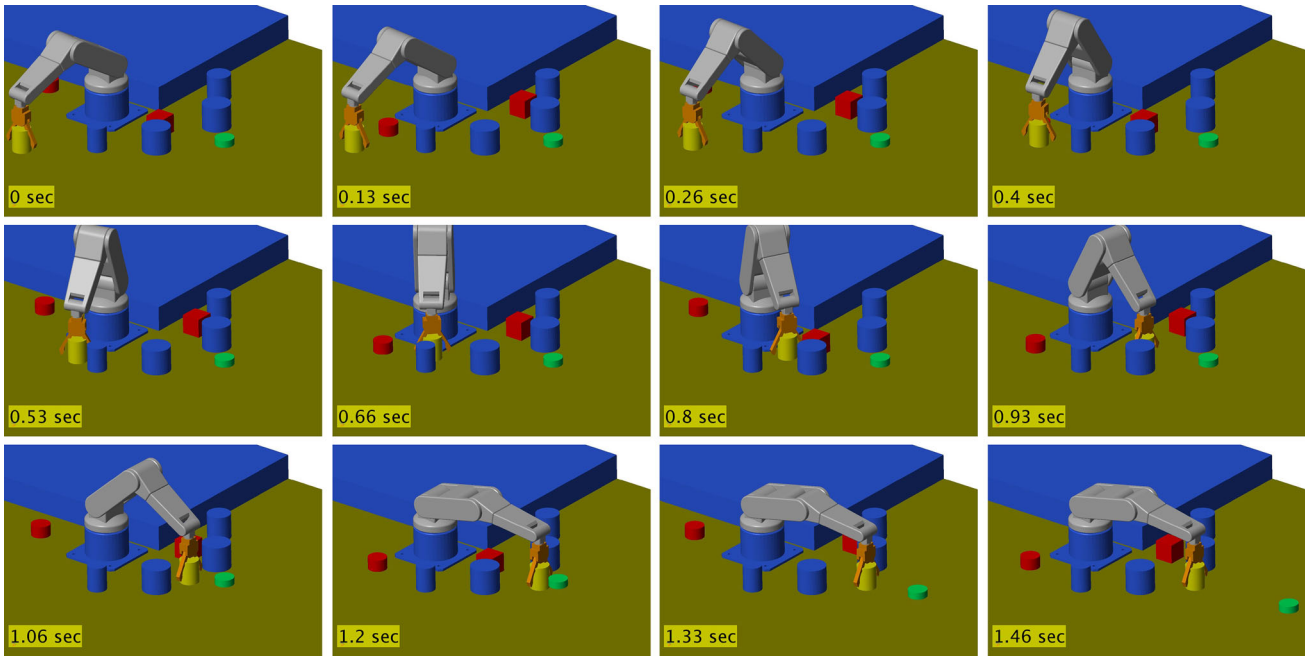


Fig. 9 The 6R manipulator's trajectory avoiding the obstacles and hitting the disk. The red obstacles are dynamic with a cyclic trajectory while the blue obstacles are static

$P_{max} = 0.05$ not to find a solution, the minimum number of random points needed is $N = 1426$ [Eq. (45)]. Therefore, by generating the N points in Σ, the probability to find a solution if one exists is 95%.

With generating N random points in Σ and using S_{max} calculated to be 3.28, the algorithm outputted 12 feasible solutions after 4.01 min. Choosing a minimum time cost function

$$H(\sigma) = \min\{t_g\}, \tag{59}$$

an optimal trajectory solution is outputted. Snapshots of the optimal motion are illustrated in Fig. 9. The end-effector navigates the grasped cylinder between the obstacles to hit the disk. The joints angle, angular velocity, and torque response of the optimal trajectory are shown in Figs. 10, 11 and 12, respectively. As could be seen, they are within the bounds defined by the kinodynamic constraints. The task response of the end-effector is presented in Fig. 13. The goal state is reached after 1.17 s while maintaining the height and pitch constraints. The results were repeatable in all attempted trials. The outputted trajectory now provides the optimal trajectory over all possible start states. In addition, for this particular start state, the trajectory is the optimal one.

The computation runtime is due to the 6×6 matrices computation of the dynamic model in (15). Removing the torque constraint (21) yields runtime of 24.17 s, that is, 89.93% less. In this simulation we have used $\Delta S = 1/3$ and separated between the constraint checks as discussed in Sect. 6.2.

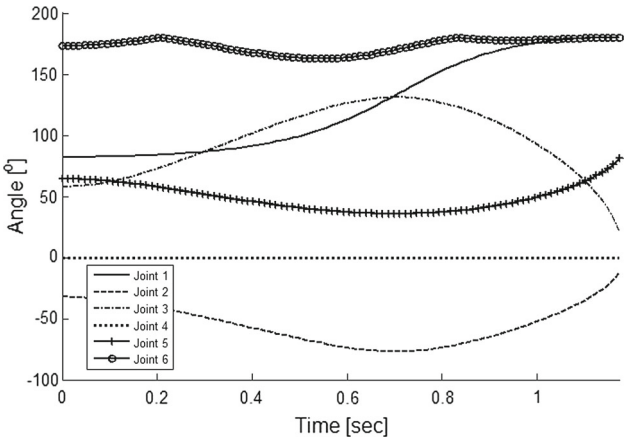


Fig. 10 The 6R joint angles within the allowed bounds

Without these methods, the runtime would excessively be 1.45 h.

We wish also to demonstrate the optimality of the SKIP solution compared to the known RRT* algorithm (Karaman and Frazzoli 2011). We tested both algorithm on the planar three degrees of freedom manipulator seen in Fig. 14 moving through dynamic obstacles and with a minimal time cost function. For that matter, an initial configuration was set and not optimized for SKIP. The joint angles, angular velocity and torque limits were set to $\pm 150°$, ± 70 RPM and ± 2 Nm, respectively. For the SKIP implementation, the desired trajectory function in the task space was chosen to be a 5th order polynomial. $N = 1061$ random points were generated with

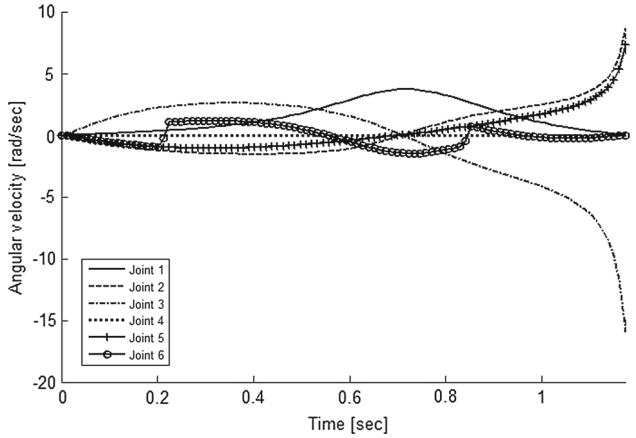


Fig. 11 The 6R joint angular velocities within the allowed bounds

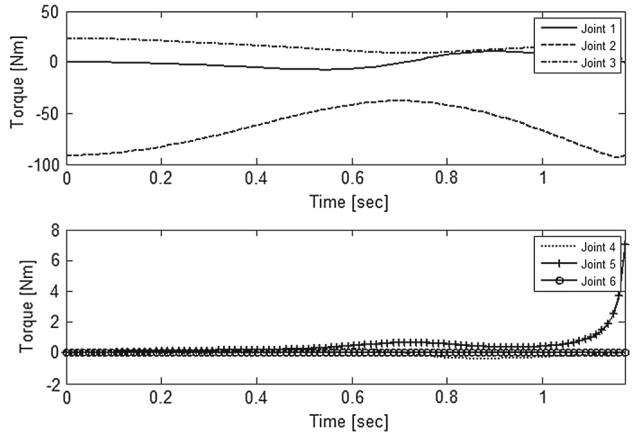


Fig. 12 The 6R joint torques within the allowed bounds

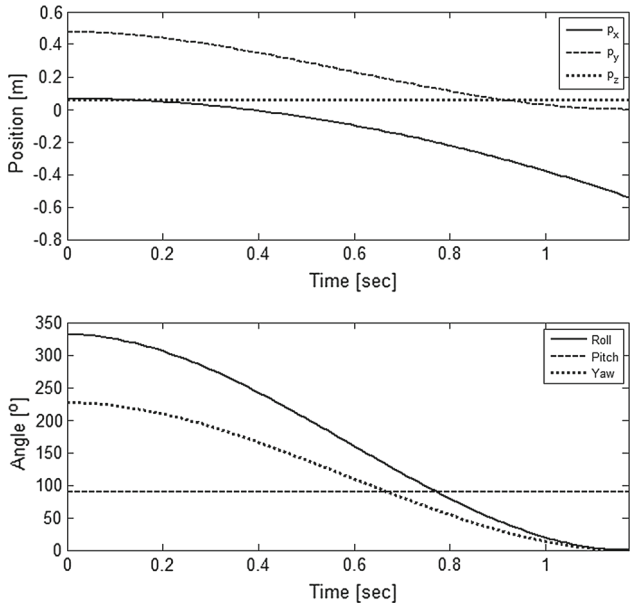


Fig. 13 The 6R end-effector task taken to the desired goal state

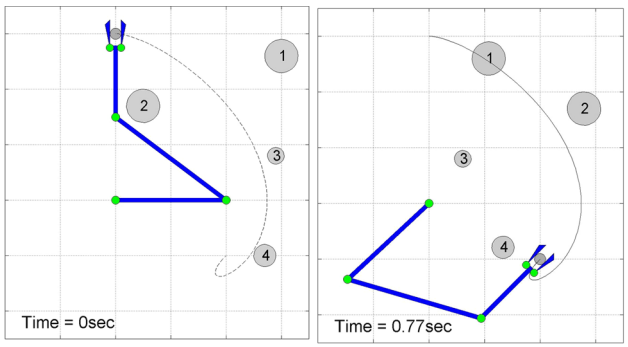


Fig. 14 A three degrees of freedom manipulator moving through dynamic obstacles from start (left) to goal (right) configurations

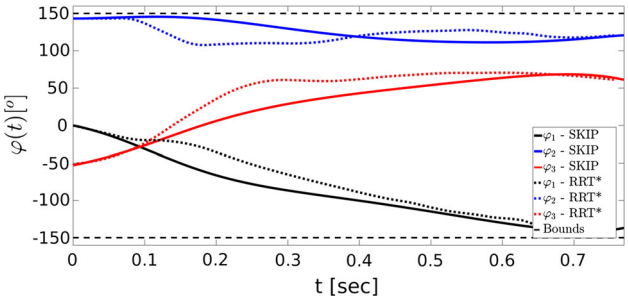


Fig. 15 The joints angles response for the 3R manipulator solved with the SKIP and RRT* algorithms (Color figure online)

98% probability of finding a solution. A time optimal solution with goal time of $t_g = 0.77$ s was found after runtime of 0.28 s. The RRT* algorithm was implemented as in Webb and van den Berg (2013) with state-time representation (Sintov and Shapiro 2014) for dynamic obstacles avoidance. The RRT* ran for 20,000 iterations generating a trajectory with $t_g = 0.756$ s. The angle response of both algorithms can be seen in Fig. 15. The SKIP output is near optimal since it is confined to a polynomial trajectory. That is, the SKIP algorithm found an optimal solution under the restrictions of a polynomial trajectory.

9 Conclusions

We present an algorithm for goal state driven trajectory optimization taking into account the kinodynamic constraints of the system. The trajectory is parameterized to a high-dimensional parameter space based on the goal state and the trajectory function. The algorithm uses the formalized constraints in the parameter space to find a feasible set of solutions. The set is found by generating random points within the allowed region of the free parameters and checking each one for feasibility. The feasibility analysis was done with an adaptive step size algorithm, decreasing the step size if the point approaches the constraint boundary. Moreover, an

optimal solution is chosen from the set of feasible solutions and a gradient descent approach is used to refine the solution to the near by local optimum. The probability to find a solution was shown to approach one if the number of random points increases to infinity. Moreover, the statistical analysis enabled an automatic parameter selection for the algorithm to find a solution in a given probability or runtime.

It was shown that the algorithm has the worst case time complexity in the order of $O(\xi N)$. It should be noticed that the algorithm plans the motion off-line prior to the motion. To reduce the runtime, the computation could be done with less strict parameters to see if solutions could be found more quickly. Future work will involve runtime reduction and application for online planning.

Currently, the allowed region Σ is user defined by preprocessing simulations. Future work will deal with the definition of the allowed region Σ. That is, finding optimal boundaries of the set Σ while decreasing the number of random points to be generated and of the runtime.

Acknowledgements The research was supported by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev.

References

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization. Berichte über verteilte messysteme*. Cambridge: Cambridge University Press.

Brooks, R. A., & Lozano-Perez, T. (1983). A subdivision algorithm configuration space for findpath with rotation. In *Proceedings of the international joint conferences on artificial intelligence* (Vol. 2, pp. 799–806).

Byravan, A., Boots, B., Srinivasa, S., & Fox, D. (2014). Space-time functional gradient optimization for motion planning. In *IEEE international conference on robotics and automation (ICRA)*

Canny, J. (1988). *The complexity of robot motion planning. ACM doctoral dissertation award*. Cambridge: MIT Press.

Cauchy, A. (1847). Méthode générale pour la résolution des systémes d'équations simultanées. *Comptes rendus de l'Académie des Sciences*, *25*, 536–538.

Clark, C. M. (2005). Probabilistic road map sampling strategies for multi-robot motion planning. *Robotics and Autonomous Systems*, *53*(3–4), 244–264.

Denny, J., Shi, K., & Amato, N. (2013). Lazy toggle PRM: A single-query approach to motion planning. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 2407–2414)

Donald, B., Xavier, P., Canny, J., & Reif, J. (1993). Kinodynamic motion planning. *Journal of the ACM*, *40*(5), 1048–1066.

Dragan, A., Ratliff, N., & Srinivasa, S. (2011). Manipulation planning with goal sets using constrained trajectory optimization. In *IEEE international conference on robotics and automation (ICRA)* (pp. 4582–4588)

Heo, Y. J., & Chung, W. K. (2013). RRT-based path planning with kinematic constraints of AUV in underwater structured environment. In *10th international conference on ubiquitous robots and ambient intelligence (URAI)* (pp. 523–525)

Hsu, D., Kindel, R., Latombe, J.-C., & Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, *21*(3), 233–255.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S. (2011). Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation (ICRA)* (pp. 4569–4574)

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894.

Kavraki, L., Svestka, P., Latombe, J. C., & Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*(4), 566–580.

Keller, H. (1976). *Numerical solution of two point boundary value problems*. Society for industrial and applied mathematics.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, *5*(1), 90–98.

Kim, D. H., Lim, S. J., Lee, D. H., Lee, J. Y., & Han, C. S. (2013) A RRT-based motion planning of dual-arm robot for (dis)assembly tasks. In *44th international symposium on robotics* (ISR) (pp. 1–6)

Kuhn, H. W., & Tucker, A. W. (1950). Nonlinear programming. In J. Neyman (Ed.), *Proceedings of the 2nd Berkeley symposium on mathematical statistics and probability* (pp. 481–492). Berkeley: University of California Press.

LaValle, S., & Kuffner J. J. J. (1999). Randomized kinodynamic planning. In *Proceedings of the IEEE international conference on robotics and automation* (Vol. 1, pp. 473–479)

Lavalle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical report.

Lebesgue, H., & May, K. (1966). *Measure and the integral*., The Mathesis Series San Francisco: Holden-Day.

Luders, B., Karaman, S., Frazzoli, E., & How, J. (2010). Bounds on tracking error using closed-loop rapidly-exploring random trees. In *American control conference (ACC)* (pp. 5406–5412)

Motonaka, K., Watanabe, K., & Maeyama, S. (2013). Motion planning of a UAV using a kinodynamic motion planning method. In *39th annual conference of the IEEE industrial electronics society (IECON)* (pp. 6383–6387)

Park, C., Pan, J., & Manocha, D. (2012). ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Proceedings of the 22nd international conference on automated planning and scheduling, ICAPS 2012* (pp. 207–215), Atibaia, São Paulo, June 25–19, 2012

Pham, Q.-C., Caron, S., & Nakamura, Y. (2013). Kinodynamic planning in the configuration space via admissible velocity propagation. In *Proceedings of robotics: Science and systems*.

Rao, A. (2014). Trajectory optimization: A survey. In H. Waschl, I. Kolmanovsky, M. Steinbuch, & L. del Re (Eds.), *Optimization and optimal control in automotive systems* (Vol. 455, pp. 3–21)., Lecture notes in control and information sciences Berlin: Springer.

Rao, A. V. (2009). A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, *135*(1), 497–528.

Ratliff, N., Zucker, M., Bagnell, J. A. D., & Srinivasa, S. (2009). Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE international conference on robotics and automation (ICRA)*

Reif, J. (1979). Complexity of the mover's problem and generalizations. In *20th annual symposium on foundations of computer science* (pp. 421–427)

Rimon, E., & Koditschek, D. (1992). Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, *8*(5), 501–518.

Schulman, J., Ho, J., Lee, A., Awwal, I., Bradlow, H., & Abbeel, P. (2013). Finding locally optimal, collision-free trajectories with

sequential convex optimization. In *Proceedings of the robotics: Science and systems (RSS)*

Schwartz, J. T., & Sharir, M. (1983). On the 'piano movers' problem. II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, *4*(3), 298–351.

Sintov, A., & Shapiro, A. (2014) Time-based RRT algorithm for rendezvous planning of two dynamic systems. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 6745–6750)

Sintov, A., & Shapiro, A. (2015). A stochastic dynamic motion planning algorithm for object-throwing. In *Proceedings of the IEEE international conference on robotics and automation*

Sintov, A., & Shapiro, A. (2017). Dynamic regrasping by in-hand orienting of grasped objects using non-dexterous robotic grippers. In *Robotics and computer-integrated manufacturing*

Song, G., & Amato, N. (2001). Randomized motion planning for car-like robots with c-prm. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems* (Vol. 1, pp. 37–42)

Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). *Robot modeling and control* (Vol. 3). New York: Wiley.

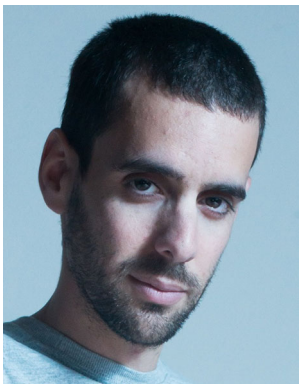
Van der Stappen, A., Halperin, D., & Overmars, M. (1993). Efficient algorithms for exact motion planning amidst fat obstacles. In *Proceedings of the IEEE international conference on robotics and automation* (Vol. 1, pp. 297–304)

Van der Stappen, A. F., Overmars, M. H., de Berg, M., & Vleugels, J. (1998). Motion planning in environments with low obstacle density. *Discrete & Computational Geometry*, *20*(4), 561–587.

Verscheure, D., Demeulenaere, B., Swevers, J., De Schutter, J., & Diehl, M. (2009). Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, *54*(10), 2318–2327.

Webb, D., & van den Berg, J. (2013). Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In: *IEEE international conference on robotics and automation (ICRA)* (pp. 5054–5061)

Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., et al. (2013). CHOMP: Covariant Hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, *32*(9–10), 1164–1193.



Avishai Sintov received his B.Sc., M.Sc., and Ph.D. degrees in Mechanical Engineering from Ben-Gurion University of the Negev, Beer-Sheva, Israel, in 2008, 2012, and 2016, respectively. He is currently a Postdoctoral research fellow in the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. His research interests include grasp planning algorithms, motion planning, dynamic manipulations, dynamic re-grasping, climbing robots, machine learning and mechanical design.