Applied Mathematical Modelling 84 (2020) 89–105

Contents lists available at ScienceDirect

Applied Mathematical Modelling

journal homepage: www.elsevier.com/locate/apm

Trajectory planning based on non-convex global optimization for serial manipulators

Shiyu Zhang[a,*], Shuling Dai[a], Andrea Maria Zanchettin[b], Renzo Villa[b]

[a] *School of Automation Science and Electrical Engineering, Beihang University Beijing 100191, China*
[b] *Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy*

ARTICLE INFO

Article history:
Received 1 June 2019
Revised 25 February 2020
Accepted 3 March 2020
Available online 10 March 2020

Keywords:
Real-time trajectory planning
Non-convex optimization
Global optimization
Machine learning
Robotics

ABSTRACT

To perform specific tasks in dynamic environments, robots are required to rapidly update trajectories according to changing factors. A continuous trajectory planning methodology for serial manipulators based on non-convex global optimization is presented in this paper. First, a kinematic trajectory planning model based on non-convex optimization is constructed to balance motion rapidity and safety. Then, a model transformation method for the non-convex optimization model is presented. In this way, the accurate global solution can be obtained with an iterative solver starting from arbitrary initializations, which can greatly improve the computational accuracy and efficiency. Furthermore, an efficient initialization method for the iterative solver based on multivariable-multiple regression is presented, which further speeds up the solution process. The results show that trajectory planning efficiency is significantly enhanced by model transformation and initialization improvement for the iterative solver. Consequently, real-time continuous trajectory planning for serial manipulators with many degrees of freedom can be achieved, which lays a basis for performing dynamic tasks in complex environments.

1. Introduction

When working in dynamic environments or interacting with humans, robots need to rapidly react to changing environments and to unexpected human actions. Therefore, the robots need to constantly adapt their motions according to the update situations. In this process, the trajectory planning problem is a bridge between motion planning and motion control, connecting path points generated by a higher-level motion planner and generating a sequence of reference configurations for lower-level controllers. In frequently changing environments, the trajectory needs to be re-planned with high frequency, which requires the calculation for each trajectory to be as fast as possible to ensure real-time performance. Furthermore, reliable solutions are required to ensure safe operation, for example, not hurting humans, colliding with other objects in shared environments or damaging the robot. Therefore, continuously solving a sequence of trajectory planning problems with high computational efficiency and reliability is an essential task.

Trajectory planning methods based on non-linear optimization are generally used to improve the motion efficiency of robots by optimizing indices such as motion time, energy and power. However, objective functions and constraints are usually complex in practical problems. When involving dynamics equations, which are of great significance for fully exploiting

* Corresponding author.
E-mail address: zhangshiyu@buaa.edu.cn (S. Zhang).

https://doi.org/10.1016/j.apm.2020.03.004
0307-904X/© 2020 Elsevier Inc. All rights reserved.

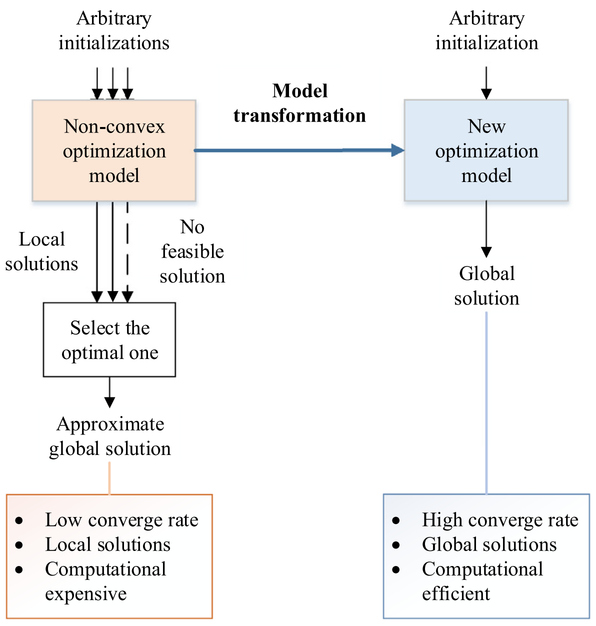

Fig. 1. Contribution of model transformation. The workflows on the left side (orange) and right side (blue) represent the traditional method and the presented method, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the inherent dynamic capabilities, but cannot always be accurately obtained and can lead to heavy computational load, the optimization problems become even more complicated and are hardly applied to on-line adaptive applications [1]. Pure kinematic optimization models are widely employed in real-time applications to reduce the computational load [2], but still are always non-convex and hard to solve precisely and efficiently and therefore cannot ensure fast and reliable solutions.

When solving non-convex optimization problems, local minimization methods tend to get stuck in local minima. Multiple restart methods and heuristic methods (such as simulated annealing and evolutionary algorithms) are widely used to find the approximate global solutions for non-convex optimization problems [3]. However, these methods are very costly and unable to guarantee the accuracy of the solution. Many researchers have adopted machine learning methods to predict an approximate initial guess for the iterative solver, which are helpful to improve the accuracy and efficiency of the solution. Unfortunately, noise appears in the database due to the inaccuracy of sample calculation. Moreover, there is also no assurance that the learned approximate initialization will lead to the global optimum. Therefore, solving the accurate global solution in real time for non-convex optimization problems is still a challenge.

In this paper, we present a methodology to efficiently and accurately solve such non-convex optimization problems and generate optimal trajectories in real time. We first construct a continuous kinematic trajectory planning model based on optimization that is suitable for applications requiring fast reactions and reliable operations in dynamic environments. Then, aiming at the non-convex optimization problem, we present two methods to improve the performance of the solutions obtained by any iterative solvers. First, we transform the non-convex model into an equivalent one, which has the same global solution but is proven to have only one local solution. In this way, the computational efficiency and accuracy can be greatly improved, as illustrated in Fig. 1. Then, we combine the model transformation method with a machine-learning-based initialization method. The new model can enhance the machine learning method and is then sped up by this method, as illustrated in Fig. 2.

We organize the remainder of this paper as follows. We first discuss state-of-the-art approaches to optimization-based trajectory planning in Section 2. Then, we describe the main problem discussed in this paper, real-time continuous trajectory planning, and construct the optimization model in Section 3. In Sections 4 and 5, we present the two methods, non-convex model transformation and a machine-learning-based solver, respectively. In Section 6, we evaluate the feasibility, accuracy and computational efficiency of the presented method using numerical experiments and validate the real-time performance. Finally, we discuss our conclusions and future work in Section 7.

2. Related work

The continuous trajectory planning problem for robots is widely used in practice, which means re-planning the trajectories for robots in real time according to the updates of the motion state of robots and the changes in the surrounding environments. It is also the foundation for various kinds of human-robot interaction applications, such as playing ball games [4], catching flying objects [5,6], collaborative manufacturing [7–9] and haptic interaction [10]. The continuous trajectory

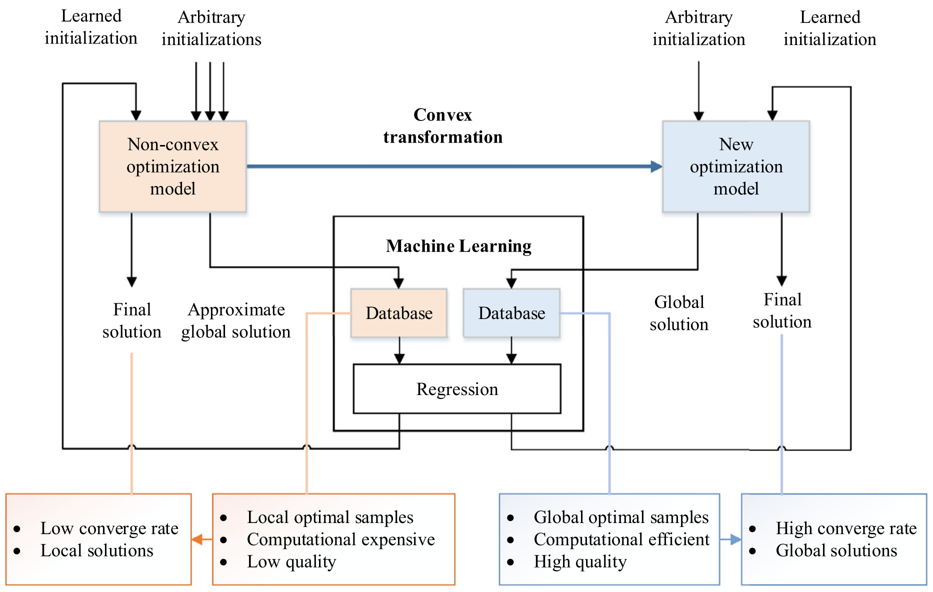

Fig. 2. Contribution of machine-learning-based initialization. The workflows on the left side (orange) and right side (blue) represent the traditional method and the presented method, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

planning problem can be split into a series of point-to-point trajectory planning problems, which are expected to be solved within a very short time period.

Trajectory planning through multiple via points is traditionally handled with interpolation-based methods [11]. In general, trajectories obtained by pure interpolation-based methods are feasible but not optimal with regard to certain performance indices. Some advanced interpolation-based methods have been presented that can generate trajectories with minimal time [12] or end-effector synthesis error [13]. However, these global interpolation-based methods are all off-line and hard to use in real-time reactive applications due to the large number of equations caused by multiple via points. An on-line straight-line trajectory generation framework was presented that can deal with different initial and final conditions and thus can be integrated into a higher-level motion planning framework and connect intermediate points [14]. However, this highly decoupled method deals only with the time minimization cost function. Many other studies have focused on optimizing multiple indices, such as time, energy and power consumption. Trajectory parameters for all degrees of freedom (DOFs) are encoded into the single non-linear optimization problem. Three separated non-linear optimization problems with criteria of minimum time, minimum energy and minimum power consumption were constructed and solved by a numerical method, combining a direct collocation and an indirect multiple shooting method [15]. Alternatively, a single optimal planning problem combining the criteria of time, energy and power consumption was built and solved by the sequential quadratic programming (SQP) method [16]. However, neither method works in real time due to the high computational complexity of non-convex optimization.

Non-convex optimizations are likely to get stuck in local minima. Multiple restart methods and heuristic methods are generally used to solve for global minima, which cannot assure the accuracy of the solutions and are too costly to operate in real time [17]. Many researchers have adopted machine learning methods to improve the computational accuracy and efficiency of global optimization for non-convex models [18]. In [19], a machine learning approach is used to predict whether a given initialization would lead to a feasible solution, emphasizing feasibility rather than optimality. Regression and prediction are used to speed up the solution of the non-convex optimization problem by first generating an optimal database off-line and then outputting good initializations [20]. However, samples are calculated with the multiple restart method, which is time consuming and unable to guarantee accuracy. A trajectory prediction method was proposed, employing the previous data to speed up the generation of new trajectories [21]. However, the samples in the database are locally optimal. This method was improved by using global optimal samples in the database [22]. In addition, a modified random restart algorithm was used to solve for the global solution, which is an order of magnitude faster than the naive solution. Furthermore, the requirements of the database and the algorithm to guarantee the performance of the solution were discussed. The probability of reaching the global minimum can be improved by increasing the density of samples and employing appropriate algorithms. However, this approach is still unable to ensure the accuracy of the solution.

Unlike previous works, we first deal with the non-convex problem model by equivalent transformation, converting the global optimization problem into a local optimization problem that can ensure the accuracy of the solution and significantly

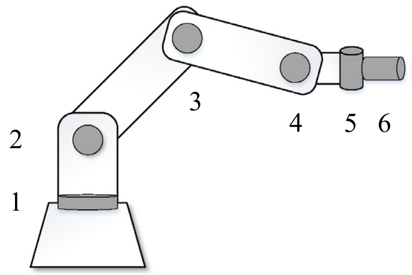

Fig. 3. Serial manipulator with rotating joints ($N_J = 6$).

Table 1
Link parameters of the manipulator and explanation.

Parameter	Meaning
α_i	Angle value from Z_i to Z_{i+1} about X_i
a_i	Perpendicular distance from Z_i to Z_{i+1} along X_i
q_i	Angle value from X_i to X_{i+1} about Z_i
d_i	Perpendicular distance from X_i to X_{i+1} along Z_i

reduce the calculation time. Furthermore, the local optimization solver is initialized by a machine learning method to further improve its computational efficiency.

3. Continuous trajectory planning based on kinematic optimization for serial manipulators

3.1. Serial manipulators

A serial manipulator is a robot arm with links serially connected by joints that can be used to manipulate objects as a human arm does.

The trajectory planning method presented in this paper is applied to a generic N_J-DOF serial manipulator with N_J rotating joints, as shown in Fig. 3. The kinematic chain of a serial manipulator is described with the Denavit Hartenberg (DH) convention [23]. $O_0 - X_0 - Y_0 - Z_0$ is the base coordinate frame, and $O_i - X_i - Y_i - Z_i (i = 1, 2, \ldots, N_J)$ are the coordinate frames for each of the N_J links. Four geometric parameters are used to describe the relationship between two adjacent joints: α_i is the link twist, a_i is the link length, q_i is the joint angle and d_i is the link offset, as illustrated in Table 1. In the structure we consider in this paper, α_i, a_i and d_i are fixed. The configuration of the robot depends only on $q_i (i = 1, 2, \ldots, N_J)$. Thus, we define the joint space variable of the serial manipulator as

$$\boldsymbol{q} = (q_1, \ldots, q_{N_J}),$$

Then, the joint space trajectory is defined as the variation of $\boldsymbol{q}$ with respect to time.

3.2. Continuous trajectory planning

In this paper, we do not consider static obstacles in the work range of robots, which is reasonable in applications such as catching flying objects [2], playing ball games [24] and performing haptic interactions [10]. Thus, we consider point-to-point trajectory planning in joint space. The trajectories can be described with parameter $\boldsymbol{C} \in \mathbb{R}^{N_C}$. Then, the joint position, velocity and acceleration are denoted as $\boldsymbol{q}(\boldsymbol{C}, t) \in \mathbb{R}^{N_J}$, $\dot{\boldsymbol{q}}(\boldsymbol{C}, t) \in \mathbb{R}^{N_J}$, and $\ddot{\boldsymbol{q}}(\boldsymbol{C}, t) \in \mathbb{R}^{N_J}$, respectively, where $t \in \mathbb{R}$ represents time. Trajectory parameter $\boldsymbol{C}$ is determined by the input variable $\boldsymbol{X} \in \mathbb{R}^{N_X}$, which includes factors such as the current motion state, the final target and environmental variables. The trajectory planning problem is essentially the mapping from input variable $\boldsymbol{X}$ to trajectory parameter $\boldsymbol{C}$

$$\boldsymbol{X} \to \boldsymbol{C} = f(\boldsymbol{X}). \tag{1}$$

When the robot works in a dynamic environment, for example, in which the objective configuration or the obstacle position changes frequently, $\boldsymbol{X}$ changes continuously as well. Thus, the trajectory must be re-planned continuously according to the updated $\boldsymbol{X}$. In the ith period, we detect the environment and update the input variable to obtain $\boldsymbol{X^i}$, then re-plan the trajectory and obtain the corresponding parameter $\boldsymbol{C^i}$, until the robot eventually reaches the final target, as shown in Fig. 4. The trajectory of the whole process is

$$\{\boldsymbol{q}(\boldsymbol{C^i}, t), t \in [(i-1)T_p, iT_p), i = 1, 2, \ldots, N_T\}$$

where N_T is the total number of planning periods and T_p represents the planning period.

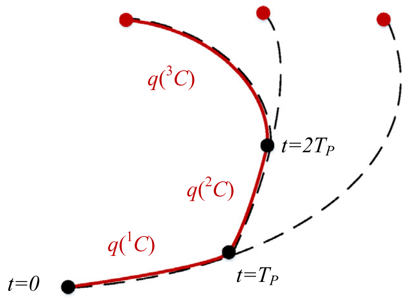

Fig. 4. Continuous trajectory planning (black lines: trajectories generated in each period; red line: the actual trajectory of the whole process). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The continuous trajectory planning problem consists of a series of point-to-point trajectory planning problems, as described in the above process. To guarantee that the robot can promptly react to the changing environment, T_p should be very small (in general less than ten milliseconds [14]). Thus, since high real-time performance is required, the calculation time for each point-to-point trajectory planning should not exceed T_p.

3.3. Trajectory parameterization

The trapezoidal velocity profile is a common trajectory representation for robots [11]. Due to the parameters with direct kinematic meaning, this profile can be analytically handled with low computational complexity. Additionally, this approach can achieve shorter motion times than higher-order trajectories. Thus, it is widely used in real-time fast reactive applications [14]. In this paper, to achieve continuous high-frequency re-planning, the trapezoidal velocity profile is selected as the trajectory representation.

The trajectory of each joint is encoded by the trapezoidal velocity profile, which consists of three phases: the uniform acceleration phase $(0-t_1)$, the maximum velocity phase (t_1-t_2) and the uniform deceleration phase (t_2-t_f). The uniform acceleration and deceleration phases have the same absolute value of acceleration, denoted as a. The velocity of the maximum velocity phase is ω_m. For each joint, the initial velocity ω_0 and the position offset $q_f=q_c-q_0$ are known, where q_c is the objective joint position and q_0 is the initial joint position. There are five variables, a, ω_m, t_1, t_2 and t_f, which are linked by three equations:

$$\frac{1}{2}\omega_m(t_f+t_2-t_1)+\frac{1}{2}\omega_0 t_1=q_c-q_0, \tag{2}$$

$$\omega_m=\omega_0+at_1, \tag{3}$$

$$\frac{\omega_m-\omega_0}{t_1}=\frac{\omega_m}{t_f-t_2}. \tag{4}$$

Define

$$\hat{H}(t,t_1,t_2)=H(t_1-t)-H(t-t_2),$$

where $H(t)$ is the Heaviside function.

Then, the trajectory of each joint is represented as

$$q(t)=q_0+\omega_0 t+\frac{1}{2}a\hat{r}(t,t_1,t_2)t^2.$$

Note that without limitations, we consider only the situations in which $q_f>0$, while ω_0 could be positive, negative or zero. In the cases of $q_f<0$, we can use the same model to calculate the trajectory parameters by accordingly changing the signs of q_f, ω_0 and ω_m.

3.4. Kinematic trajectory planning model based on optimization

In this paper, we emphasize the reaching process in dynamic situations, in which the robot moves from the current state to the constantly changing target state and has to keep updating the trajectory to ensure safe and reliable operation. In these applications, achieving successful task execution is more important than fully exploiting the dynamics property. Thus, we choose a kinematic trajectory planning model. This approach sacrifices some dynamical optimality but still always ensures the dynamical feasibility by conservatively choosing the maximum allowed accelerations to limit torques.

In the continuous re-planning problem, we consider the point-to-point trajectory planning of each period, which returns a trajectory from the current configuration to the target according to the new updated state. When assuming a N_J-DOF robot, the predicted objective configuration serves as the target, denoted as $\boldsymbol{q_c} \in \mathbb{R}^{N_J}$. The current configuration and velocity serve as the initial configuration and velocity, denoted as $\boldsymbol{q_0} \in \mathbb{R}^{N_J}$ and $\boldsymbol{\omega_0} \in \mathbb{R}^{N_J}$, respectively. In the kinematic trajectory planning problem, we consider only the initial velocity $\boldsymbol{\omega_0}$ and the offset $\boldsymbol{q_f} = \boldsymbol{q_c} - \boldsymbol{q_0}$, rather than the individual $\boldsymbol{q_c}$ and $\boldsymbol{q_0}$. Therefore, the input variable is defined as $\boldsymbol{X} = (\boldsymbol{q_f}, \boldsymbol{\omega_0}) \in \mathbb{R}^{2N_J}$.

Trajectory variables of the ith joint are denoted as a^i, ${\omega_m}^i$, ${t_1}^i$, ${t_2}^i$ and ${t_f}^i$. All the joints are required to move simultaneously, i.e., to have the same motion time, ${t_f}^1 = {t_f}^2 = \cdots = {t_f}^{N_J} = t_f$.

To achieve safety and rapidity of robot motion, we specify Eq. (1) as an optimization problem:

$$\boldsymbol{X} \to \boldsymbol{C} = \arg\min_{\boldsymbol{C}} F(\boldsymbol{X}), \tag{5}$$

in which F is the objective function and the trajectory parameter $\boldsymbol{C}$ is specified with the trajectory variables.

On the one hand, to ensure safety, the robot is expected to work gently with low accelerations. On the other hand, to achieve motion rapidity, the motion time is expected to be as short as possible. Unfortunately, there is a conflict between safety and rapidity. Therefore, we make a compromise by weighing the two aspects and define the objective function as

$$F(\boldsymbol{C}) = \sum_{i=1}^{N_J} \alpha^i \left(\frac{a^i}{{a_{max}}^i} \right)^2 + \alpha^{N_J+1} \left(\frac{t_f}{t_{max}} \right)^2, \tag{6}$$

where t_{max} is the maximum allowable motion time, ${a_{max}}^i$ is the maximum allowable acceleration of the ith joint, and α^i is the weighting coefficient, which satisfies

$$\sum_{i=1}^{N_J+1} \alpha^i = 1.$$

According to the mechanism limitations, trapezoidal velocity profiles and requirements of the specific task, the ranges of the five variables associated with the ith joint are as follows:

$$0 \le a^i \le {a_{max}}^i \tag{7}$$

$$\max(0, {\omega_0}^i) \le {\omega_m}^i \le {\omega_{max}}^i \tag{8}$$

$$0 \le {t_1}^i \le t_f/2, \tag{9}$$

$${t_1}^i \le {t_2}^i \le t_f, \tag{10}$$

$$0 \le t_f \le t_{max}, \tag{11}$$

where inequalities (9) and (10) can be simplified to

$${t_1}^i \le {t_2}^i,$$

which is equivalent to

$$2{\omega_m}^i - {\omega_0}^i - a^i t_f \le 0. \tag{12}$$

3.5. Practical applications

This model can be applied in many practical applications. Requirements of specific tasks can be transformed into kinematic variables (position, velocity, acceleration and motion time) and encoded in the optimization problem (5) with the input variable, the objective function and the constraints.

This approach is particularly suitable for applications in which robots rapidly detect and react to changing environments to successfully execute tasks or guarantee safety, such as the reaching process of human-robot interaction [25] and catching objects in flight [2]. We give two examples to illustrate how the method works in practical cases: human-robot interaction and moving obstacle avoidance.

We consider a class of human–robot interaction applications in which robots and humans work in a shared workspace and interact with each other to perform tasks such as hand-over [26], collaborative assembly [27], and haptic feedback [10], as shown in Fig. 5. In the reaching process, the system keeps detecting the human motion and accordingly predicting new targets. The target state is encoded within parameter $\boldsymbol{X}$ in Eq. (5). The safety requirement can be transformed into position and velocity constraints by specifying the target configuration $\boldsymbol{q_c}$ and the maximal allowed velocity $\boldsymbol{\omega}_{max}$ (applying

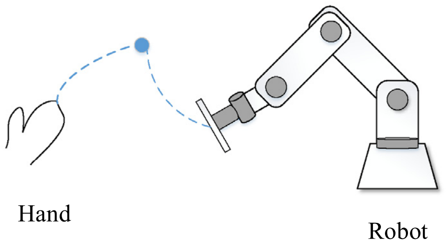

Fig. 5. Human-robot interaction.

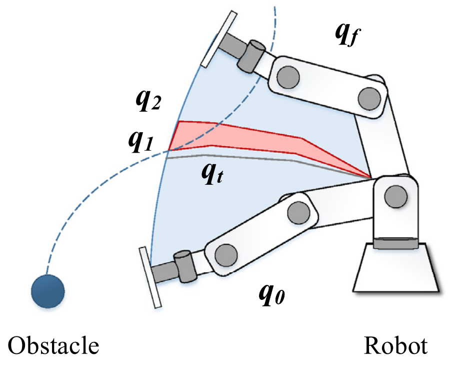

Fig. 6. Moving obstacle detection and avoidance.

constraint (8)) [10]. The reaction rapidity requirement can be encoded by motion time constraints (specifying the maximal motion time t_{max} in constraint (11)) [20].

This model can also be used in obstacle avoidance when the robot is working in a dynamic environment with moving obstacles. The motion of the obstacle is detected, and the trajectory is predicted. When the robot is moving, if a collision with any moving obstacle is predicted, we can adjust the time evolution of the original path of the robot to avoid collision [28], specifically, slowing down the robot by setting the intermediate points along the original path as temporal goals (updating $\boldsymbol{X}$). For example, as shown in Fig. 6, while the robot is moving from the initial configuration $\boldsymbol{q_0}$ to the final configuration $\boldsymbol{q_f}$, an obstacle is moving in the workspace, whose predicted trajectory overlaps with the area that will be swept by the robot (represented by the blue shadow). The configurations that allow possible collision with the obstacles are from $\boldsymbol{q_1}$ to $\boldsymbol{q_2}$. The swept area is represented by the red shadow. Before the obstacle passes the red area, an intermediate configuration before $\boldsymbol{q_1}$, represented as $\boldsymbol{q_t}$, is selected as the temporal goal with which the robot is expected to stop moving. After the obstacle has passed the red area, the goal is set back to the final goal $\boldsymbol{q_f}$. If the obstacle passes through the red area before the robot reaches $\boldsymbol{q_t}$, the robot will not actually stop in the whole process; instead, it will first slow down and then speed up. If the obstacle has not passed through the red area when the robot reaches $\boldsymbol{q_t}$, the robot will stop at $\boldsymbol{q_t}$ and wait for the obstacle to pass. In this way, we can ensure that the robot never collides with the obstacle.

4. Non-convex optimization model transformation

4.1. Overview

According to Section 3.4, in the point-to-point trajectory planning problem of each period, for each DOF, there are five variables (a, ω_m, t_1, t_2 and t_f) and three equality constraints (Eqs. (2)–(4)). To construct the optimization problem model, some of the variables (not less than 2) are selected as the optimization parameters, while others can be determined by the equality constraints. The inequality constraints are derived by the variable limitations (inequalities (7), (8), (11) and (12)) accordingly. Problem models with different properties and complexities can be obtained by selecting different optimization parameters.

A straightforward method is to select a and t_f as the optimization parameters [2] (referred to as Model 1). In this way, the objective function is simple and intuitive, producing a balance of small acceleration and motion time. However, the inequality constraints are very complicated and highly non-convex, which leads to a non-convex optimization problem with multiple local minimizers. The multiple restart method is a general way to find an approximate global solution. It contains a local phase, in which the multiple local searches are run to yield candidates of the global minimum, and a global phase, in which the function values of the local solutions are evaluated to find the best one. However, this local-global process is time demanding and cannot guarantee the accuracy [17]. The detailed analyses are described in Section 4.2.

To improve the feasibility and efficiency of calculation, we make an equivalent transformation by parameter substitution, by which the convexity of the optimization problem can be changed but the solution of the problem remains the same [29]. We select ω_m and t_f as the optimization parameters instead of a and t_f and accordingly modify the constraints (referred to as Model 2). Then, a new optimization model is derived, which has a unique local solution. That means that the global solution can be found by running only the local search phase till finding a feasible local solution. By this way, the calculating time for multiple local searches and global evaluation can be saved and the global solution is guaranteed. Thus, the solution accuracy and efficiency are significantly improved by variable substitution and model transformation, as illustrated in Fig. 1. The detailed analyses and proof are described in Section 4.3.

4.2. Model 1: Selecting a^{i} $(i = 1, \ldots, N_J)$ and t_{f} as Optimization Parameters

Considering a N_J-DOF serial robot, there are $N_J + 1$ optimization parameters:

$$\boldsymbol{C} = (a^1, \ldots, a^{N_J}, t_f) \in \mathbb{R}^{N_J+1}.$$

Then, the objective function is described as Eq. (6), which is an obvious convex function.

The ranges of the optimization parameters are determined by inequalities (7) and (11).

According to Eqs. (2)–(4), we obtain

$$\omega_m{}^i = \frac{(a^i t_f + \omega_0{}^i) - \sqrt{(a^i t_f + \omega_0{}^i)^2 - 2(\omega_0{}^i)^2 - 4q_f{}^i a^i}}{2}. \tag{13}$$

Substituting Eq. (13) into constraints (8) and (12), we obtain extremely complicated constraints due to the square root in the expression (Appendix A). This optimization problem is highly non-convex and thus has multiple local minimizers.

4.3. Model 2: Selecting $\omega_\mathrm{m}{}^{\mathrm{i}}$ $(i = 1, \ldots, N_J)$ and t_{f} as Optimization Parameters

Considering a N_J-DOF serial robot, there are $N_J + 1$ optimization parameters:

$$\boldsymbol{C} = (\omega_m{}^1, \ldots, \omega_m{}^{N_J}, t_f) \in \mathbb{R}^{N_J+1}.$$

According to Eqs. (2)–(4), we obtain

$$a^i = \frac{(\omega_m{}^i)^2 + (\omega_0{}^i)^2/2 - \omega_0{}^i\omega_m{}^i}{\omega_m{}^i t_f - q_f{}^i}. \tag{14}$$

The objective function is described as Eq. (6) together with (14).

The ranges of optimization parameters are determined by inequalities (8) and (11). Substituting Eq. (14) into inequalities (7) and (12), we can derive the inequality constraints.

In this way, we reformulate the optimization problem as the new form, denoted as P:

$$P: \quad \boldsymbol{C} = (\omega_m{}^1, \ldots, \omega_m{}^{N_J}, t_f) \in \mathbb{R}^{N_J+1} \tag{15a}$$

$$\omega_m{}^i \in [\max(0, \omega_0{}^i), \omega_{max}{}^i] \tag{15b}$$

$$t_f \in [0, t_{max}] \tag{15c}$$

$$F(\boldsymbol{C}) = \sum_{i=1}^{N_J} \alpha^i \left(\frac{a^i}{a_{max}{}^i}\right)^2 + \alpha^{N_J+1}\left(\frac{t_f}{t_{max}}\right)^2 \tag{15d}$$

$$a^i = \frac{(\omega_m{}^i)^2 + (\omega_0{}^i)^2/2 - \omega_0{}^i\omega_m{}^i}{\omega_m{}^i t_f - q_f{}^i} \tag{15e}$$

$$q_f{}^i - \omega_m{}^i t_f < 0 \tag{15f}$$

$$a^i - a_{max}{}^i \leq 0 \tag{15g}$$

$$2\omega_m{}^i - \omega_0{}^i - a^i t_f \leq 0 \tag{15h}$$

Theorem 1. *The optimization problem P has a unique local minimizer, which is the global minimizer.*

The objective function of P, represented by Eqs. (15d) and (15e), is convex in the domain. constraints (15f) and (15g) are convex, while constraint (15h) is non-convex. Thus, P is not a convex optimization problem. However, for each non-convex problem P, we can find an equivalent convex optimization problem P' by dealing with the non-convex constraint. Thus, P has a unique local solution, which is the global solution.

Proof. We first determine the convexity of the objective function and each constraint, then convert the original problem P into P', and finally prove that P' is a convex optimization problem.

(1) The objective function is convex in the domain

To simplify the expression, the objective function is reformulated as

$$F({\omega_m}^1,\ldots,{\omega_m}^{N_J},t_f)=\sum_{i=1}^{N_J}\mu^i(a^i)^2+\mu^{N_J+1}{t_f}^2, \tag{16}$$

where $\mu^i(i=1,\ldots,N_J+1)$ are weighting coefficients satisfying

$$\mu^i\geq 0.$$

Define

$$h_i({\omega_m}^1,\ldots,{\omega_m}^{N_J},t_f)=a^i(i=1,\ldots,N_J),$$

$$h_{N_J+1}({\omega_m}^1,\ldots,{\omega_m}^{N_J},t_f)=t_f.$$

Then, the Hessian matrix of h_1 is

$$\triangledown^2 h_1=\begin{pmatrix}\dfrac{\partial^2 a^1}{\partial({\omega_m}^1)^2} & \cdots & \dfrac{\partial^2 a^1}{\partial{\omega_m}^1\partial{\omega_m}^{N_J}} & \dfrac{\partial^2 a^1}{\partial{\omega_m}^1\partial t_f}\\ \vdots & \ddots & \vdots & \vdots\\ \dfrac{\partial^2 a^1}{\partial\omega_{m,N_J}\partial{\omega_m}^{N_J}} & \cdots & \dfrac{\partial^2 a^1}{\partial({\omega_m}^{N_J})^2} & \dfrac{\partial^2 a^1}{\partial{\omega_m}^{N_J}\partial t_f}\\ \dfrac{\partial^2 a^1}{\partial t_f\partial{\omega_m}^1} & \cdots & \dfrac{\partial^2 a^1}{\partial t_f\partial{\omega_m}^{N_J}} & \dfrac{\partial^2 a^1}{\partial{t_f}^2}\end{pmatrix}.$$

According to Eq. (14), we have

$$\frac{\partial^2 a^i}{\partial({\omega_m}^i)^2}=\frac{[({q_c}^i-{\omega_0}^i t_f)^2+{q_c}^i]({\omega_m}^i)^{-3}}{t_f-{q_c}^i({\omega_m}^i)^{-1}}>0,$$

and

$$\frac{\partial^2 a^i}{\partial{\omega_m}^i\partial{\omega_m}^j}=0,\quad \frac{\partial^2 a^i}{\partial({\omega_m}^j)^2}=0,\quad \frac{\partial^2 a^i}{\partial t_f\partial{\omega_m}^j}=0\quad(i\neq j).$$

Then, we obtain

$$\triangledown^2 h_1=\begin{pmatrix}\dfrac{\partial^2 a^1}{\partial({\omega_m}^1)^2} & \cdots & 0 & \dfrac{\partial^2 a^1}{\partial{\omega_m}^1\partial t_f}\\ \vdots & \ddots & \vdots & \vdots\\ 0 & \cdots & 0 & 0\\ \dfrac{\partial^2 a^1}{\partial t_f\partial{\omega_m}^1} & \cdots & 0 & \dfrac{\partial^2 a^1}{\partial{t_f}^2}\end{pmatrix}. \tag{17}$$

The leading principal minors are

$$D_1=\frac{\partial^2 a^1}{\partial({\omega_m}^1)^2}>0,\quad D_i=0(i=2,\ldots,N_J+1).$$

Thus, $\triangledown^2 h_1$ is positive semidefinite. Similarly, $\triangledown^2 h_i$'s are all positive semidefinite. Consequently, $h_i({\omega_m}^1,\ldots,{\omega_m}^{N_J},t_f)$ is convex in the domain.

Similarly, the Hessian matrix of h_{N_J+1}

$$\nabla^2 h_{N_J+1} = \begin{pmatrix} \frac{\partial^2 t_f}{\partial ({\omega_m}^1)^2} & \dots & \frac{\partial^2 t_f}{\partial {\omega_m}^1 \partial {\omega_m}^{N_J}} & \frac{\partial^2 t_f}{\partial {\omega_m}^1 \partial t_f} \\ \vdots & \ddots & \vdots & \vdots \\ \frac{\partial^2 t_f}{\partial {\omega_m}^{N_J} \partial {\omega_m}^1} & \dots & \frac{\partial^2 t_f}{\partial ({\omega_m}^{N_J})^2} & \frac{\partial^2 t_f}{\partial {\omega_m}^{N_J} \partial t_f} \\ \frac{\partial^2 t_f}{\partial t_f \partial {\omega_m}^1} & \dots & \frac{\partial^2 t_f}{\partial t_f \partial {\omega_m}^{N_J}} & \frac{\partial^2 t_f}{\partial {t_f}^2} \end{pmatrix} = \begin{pmatrix} 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

is positive semidefinite, and consequently, $h_{N_J+1}({\omega_m}^1, \dots, {\omega_m}^{N_J}, t_f)$ is convex.

Define

$$g(x) = x^2,$$

which is convex and non-decreasing when $x > 0$. Since $h_i (i = 1, \dots, N_{J+1})$ is convex in the domain, according to the composition rule, $g(h_i)$ is convex.

Since $\mu^i \geq 0$, according to the sum rule, $\sum_{i=1}^{N_J+1} \mu^i g(h_i)$ is convex.

In conclusion, the objective function $F(\boldsymbol{C})$ is convex in the domain.

(2) Convex constraints

We determine the convexity of constraints (15f) and (15g).

Inequality (15f) is equivalent to

$$t_f > \frac{{q_f}^i}{{\omega_m}^i}.$$

Define

$$y_{1i}({\omega_m}^1, \dots, {\omega_m}^{N_J}) = \frac{{q_f}^i}{{\omega_m}^i}.$$

The Hessian matrix of y_{11}

$$\nabla^2 y_{11} = \begin{pmatrix} \frac{\partial^2 y_{11}}{\partial ({\omega_m}^1)^2} & \dots & \frac{\partial^2 y_{11}}{\partial {\omega_m}^1 \partial {\omega_m}^{N_J}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 y_{11}}{\partial {\omega_m}^{N_J} \partial {\omega_m}^1} & \dots & \frac{\partial^2 y_{11}}{\partial ({\omega_m}^{N_J})^2} \end{pmatrix} = \begin{pmatrix} 2{q_f}^1 ({\omega_m}^1)^{-3} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}$$

is positive semidefinite. Thus, $y_{11}({\omega_m}^1, \dots, {\omega_m}^{N_J})$ is convex, and its epigraph

$$S_{11} = \{\boldsymbol{C} | t_f > y_{11}\}$$

is a convex set.

Similarly, for $i = 2, \dots, N_J$, $S_{1i} = \{\boldsymbol{C} | t_f > y_{1i}\}$ is convex.

Inequality (15f) is equivalent to

$$t_f \geq \frac{{\omega_m}^i - {\omega_0}^i + \left[\frac{({\omega_0}^i)^2}{2} + {a_{max}}^i {q_f}^i\right]({\omega_m}^i)^{-1}}{{a_{max}}^i}.$$

Define

$$y_{2i}({\omega_m}^1, \dots, {\omega_m}^{N_J}) = \frac{{\omega_m}^i - {\omega_0}^i + \left[\frac{({\omega_0}^i)^2}{2} + {a_{max}}^i {q_f}^i\right]({\omega_m}^i)^{-1}}{{a_{max}}^i}.$$

The Hessian matrix of y_{21}

$$\nabla^2 y_{21} = \begin{pmatrix} \frac{\partial^2 y_{21}}{\partial ({\omega_m}^1)^2} & \dots & \frac{\partial^2 y_{21}}{\partial {\omega_m}^1 \partial {\omega_m}^{N_J}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 y_{21}}{\partial {\omega_m}^{N_J} \partial {\omega_m}^1} & \dots & \frac{\partial^2 y_{21}}{\partial ({\omega_m}^{N_J})^2} \end{pmatrix} = \begin{pmatrix} \left(\frac{({\omega_0}^1)^2}{{a_{max}}^1} + 2{q_c}^1\right)({\omega_m}^1)^{-3} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}$$

is positive semidefinite. Thus, $y_{21}({\omega_m}^1, \dots, {\omega_m}^{N_J})$ is convex, and its epigraph

$$S_{21} = \{\mathbf{C} | t_f \geq y_{21}\}$$

is a convex set.

Similarly, for $i = 2, \dots, N_J$, $S_{2i} = \{\mathbf{C} | t_f \geq y_{2i}\}$ is convex.

(3) Non-convex constraints

We determine the convexity of constraint (15h).

Inequality (15h) is equivalent to

$$t_f \geq -2{q_f}^i \frac{2{\omega_m}^i - {\omega_0}^i}{2({\omega_m}^i)^2 - ({\omega_0}^i)^2}.$$

Define

$$y_{3i}({\omega_m}^1, \dots, {\omega_m}^{N_J}) = 2{q_f}^i \frac{2{\omega_m}^i - {\omega_0}^i}{2({\omega_m}^i)^2 - ({\omega_0}^i)^2}.$$

The Hessian matrix of y_{31}

$$\triangledown^2 y_{31} = \begin{pmatrix} \frac{\partial^2 y_{31}}{\partial ({\omega_m}^1)^2} & \cdots & \frac{\partial^2 y_{31}}{\partial {\omega_m}^1 \partial {\omega_m}^{N_J}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 y_{31}}{\partial {\omega_m}^{N_J} \partial {\omega_m}^1} & \cdots & \frac{\partial^2 y_{31}}{\partial ({\omega_m}^{N_J})^2} \end{pmatrix} = \begin{pmatrix} \frac{2(2{\omega_m}^1 - {\omega_0}^1)^3 + 2({\omega_m}^1)^3 + ({\omega_0}^1)^3}{(2{\omega_m}^1 - {\omega_0}^1/2)^3} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$$

is positive semidefinite. Thus, $-y_{31}({\omega_m}^1, \dots, {\omega_m}^{N_J})$ is concave, and its epigraph

$$S_{31} = \{\mathbf{C} | t_f \geq -y_{31}\}$$

is not a convex set.

Similarly, for $i = 2, \dots, N_J$, $S_{3i} = \{\mathbf{C} | t_f \geq -y_{3i}\}$ is not convex.

(4) Problem transformation

First, we consider only the convex constraints (constraints (15f) and (15g)). Since S_{1i} and S_{2i} are both convex, the intersection

$$S' = \bigcap_{i=1}^{N_J} (S_{1i} \bigcap S_{2i})$$

is convex.

Then, we deal with the non-convex constraint (constraint (15h)). Assuming the ith component is effective, we obtain

$$2{\omega_m}^i - {\omega_0}^i - a^i t_f = 0 \tag{18}$$

Solving the systems of equations of (18) and (15e), we can derive

$$a^i = \frac{({\omega_m}^i)^2 - ({\omega_0}^i)^2/2}{{q_f}^i},$$

which is substituted into the objective function (16) to obtain the new objective function F'. Then, the original problem P is converted into a new problem P', which leads to optimizing F' on S'.

If the ith component is effective, we modify the expression of a^i in $\triangledown^2 h_i$ (Eq. (17)). Then, in $\triangledown^2 h_i$, we have

$$\frac{\partial^2 a^i}{\partial ({\omega_m}^i)^2} = \frac{4}{({q_f}^i)^2} \left[3({\omega_m}^i)^2 - \frac{({\omega_0}^i)^2}{2} \right] > 0$$

Thus, $\triangledown^2 h_i$ is positive semidefinite, and consequently, $h_i({\omega_m}^1, \dots, {\omega_m}^{N_J}, t_f)$ is convex in the domain.

Therefore, F' is convex in the domain. P' is a convex optimization problem and has a unique local minimizer, which is the global minimizer.

Consequently, the original problem P has a unique local minimizer. □

5. Real-time solver based on machine learning

5.1. Overview

With the convex transformation of the optimization model described in Section 4, the global solution can be solved with a local iterative solver starting from an arbitrary initial guess, which is denoted as

$$\boldsymbol{C}^* = O_{\boldsymbol{C_0}}(\boldsymbol{X}), \tag{19}$$

where $\boldsymbol{C_0}$ is the initial guess and $\boldsymbol{C}^*$ is the global minimizer.

Without the multiple restart process, the amount of calculation has been significantly reduced. A good initialization that is close to the accurate solution is helpful to reduce the iteration times and further speed up the solution process.

A machine learning method is applied in this paper to select good initializations. A set of parameters approximating the accurate solution is obtained by learning from the previous data, which serves as the initial guess for the iterative solver to obtain the optimal parameter. The detailed procedure is as follows.

1. Generating a database
   We select N_D groups of input variables $\boldsymbol{X^i}(i = 1, 2, \ldots, N_D)$ and solve the optimal parameters by Eq. (19)
   $$\boldsymbol{C}^{*i} = O(\boldsymbol{X^i}).$$
   Then, the optimal database is constructed as
   $$D = \{\boldsymbol{X^i}, \boldsymbol{C}^{*i} | i = 1, 2, \ldots, N_D\},$$
   which can be either artificially generated off-line or obtained by accumulating the previous motion data.
2. Establishing regression model
   By analyzing and processing the data in D, the mapping from input variable $\boldsymbol{X}$ to optimal parameter $\boldsymbol{C}^*$ is established:
   $$\boldsymbol{X} \to \boldsymbol{C}^* = R(\boldsymbol{X}).$$
   The regression model can be established both on-line and off-line, which depends on the specific regression method.
3. Learning the initial guess
   According to a new input variable $\boldsymbol{X'}$, an approximate optimal parameter $\boldsymbol{C'}$ is obtained with the regression model in Step 2
   $$\boldsymbol{C'} = R(\boldsymbol{X'}).$$
4. Solving the global optimal parameter
   Serving the approximate parameter $\boldsymbol{C'}$ obtained in Step 3 as the initial guess for the iterative solver, we can obtain the optimal parameter $\boldsymbol{C}^*$
   $$\boldsymbol{C}^* = O_{\boldsymbol{C'}}(\boldsymbol{X'}).$$

With the combination of the model transformation, the performance of the machine learning framework can be greatly enhanced, as illustrated in Fig. 2. In Step 1, since the database is filled with global optimal samples rather than local ones, better database quality is obtained, which therefore helps build a better regression model in Step 2 and obtain a better initialization in Step 3. Similarly, Step 4 always outputs the global solution. In return, a better initialization obtained in Step 3 makes the iterative process even more efficient.

5.2. Multivariable-multiple regression

We consider the regression model

$$\boldsymbol{C} = R(\boldsymbol{X}),$$

where $\boldsymbol{X} \in \mathbb{R}^{N_X}$ is the input variable and $\boldsymbol{C} \in \mathbb{R}^{N_C}$ is the output variable.

Regression models are required to output multiple dimensional parameters. However, traditional regression methods generally can build the mapping from the input variable to the single dimensional output only. To solve this problem, multivariable-multiple regression is presented in this paper, where multivariable means multiple dimensional inputs and multiple means multiple dimensional outputs.

First, a regression model $R^{(i)}$ for each element of output variable $C^{(i)}$ is built as

$$C^{(i)} = R^{(i)}(\boldsymbol{X}).$$

As each $R^{(i)}$ is built independently, the component of $\boldsymbol{X}$, which has a major effect on $C^{(i)}$, may be different. Thus, we can select a specific component of $\boldsymbol{X}$ as the feature vector for each $R^{(i)}$, denoted as $\boldsymbol{X_F}^{\boldsymbol{(i)}}$.

For each $C^{(i)}$, we select $\boldsymbol{X_F}$ as the feature vector and build the regression model

$$\boldsymbol{X} \to C^{(i)} = R^{(i)}(\boldsymbol{X_F}^{\boldsymbol{(i)}}).$$

Consequently, the multivariable-multiple regression model is

$$\boldsymbol{R} = \{R^{(i)} | i = 1, 2, \ldots, N_C\}.$$

Table 2
Comparisons of the two optimization models.

Index	Model 1	Model 2
Feasible success rate	0.359	1.000
Global rate	0.061	1.000
Calculation time (s)	1.359	0.173

5.3. Regression methods

For a new input variable $\boldsymbol{X}'$, an initial parameter $\boldsymbol{C}'$ is predicted by the regression model. Different regression methods can be adopted to build mappings between $\boldsymbol{X}$ and $\boldsymbol{C}$. Three methods, k-nearest neighbours regression (k-NN), support vector machine regression (SVR) and Gaussian process regression (GPR) [30], are mainly discussed in this paper and are further detailed in Section 6.2.2.

6. Results

First, in Section 6.1, we verified the improvement of solution feasibility, accuracy and computational efficiency by the model transformation method described in Section 4. Then, in Section 6.2, we evaluated the further feasibility and computational efficiency enhancement by combining the machine learning method described in Section 5. Finally, in Section 6.3, we present two cases to show the usage of this method in practical applications described in Section 3.5.

6.1. Model transformation

We conducted numerical experiments to verify the feasibility, accuracy and efficiency improvement in the iterative solution process with the model transformation method introduced in Section 4.

Considering a robot with $N_J = 6$, we randomly selected 1000 groups of inputs $\boldsymbol{X} = (\boldsymbol{q_f}, \boldsymbol{\omega_0})$ in the work range. Then, we employed the SQP algorithm to iteratively solve the optimization problems (5) with the two optimization models introduced in Section 4.2 (Model 1) and Section 4.3 (Model 2).

We implemented the algorithm in MATLAB on a laptop equipped with an Intel Core i7-6700HQ 2.6 GHz CPU with 16 GB RAM using a single core. We evaluated the feasibility, accuracy and computational efficiency and made comparisons of the two optimization models.

The feasibility is evaluated by the success rate of obtaining a feasible solution. Assuming N_{total} groups of optimization specified by different inputs are run, in which $N_{success}$ groups converge to the feasible solutions, the feasible success rate is

$$R_s = \frac{N_{success}}{N_{total}}.$$

The accuracy indicates whether the final solution is the global solution and is evaluated by the rate of obtaining the global solution. In the $N_{success}$ feasible solutions, if N_{global} of them are global solutions, the global rate is

$$R_g = \frac{N_{global}}{N_{success}}.$$

The computational efficiency is evaluated by the calculation time of solving each optimization problem.

With Model 1, we ran the SQP solver with 10 randomly selected initializations and selected the optimal local solution (if any existed) as the approximate global solution. With Model 2, we ran the same SQP solver with randomly selected initializations. The solver was restarted with other initializations only if no feasible solution had been found. The maximum attempt time was 10.

The results are shown in Table 2. Very poor feasibility and accuracy were achieved for Model 1 with 10 restarts. Only 359 of the 1000 optimization problems converged to feasible solutions, and 22 of them converged to the global solutions. Moreover, a long calculation time was required due to the multiple restarts with different initializations. The feasibility and accuracy were greatly improved after model transformation. With a maximum restart time of 10, all 1000 groups of optimization problems converged to feasible solutions, and all of them were global solutions. As a consequence, the calculation time was greatly reduced with the reduction of restart times.

6.2. Real-time solver based on machine learning

As illustrated in Section 5, adopting the machine learning method to select good initializations can further improve the feasibility and reduce the on-line calculation time. We conducted multivariable multiple regression with different feature selections and regression methods and evaluated the performance improvements.

Considering the same robot used in Section 6.1, we randomly selected 8000 groups of inputs $\boldsymbol{X} = (\boldsymbol{q_f}, \boldsymbol{\omega_0})$ in the working range. Then, we adopted Model 2 with the same solver mentioned in Section 6.1 to calculate the optimal parameters $\boldsymbol{C}$ and

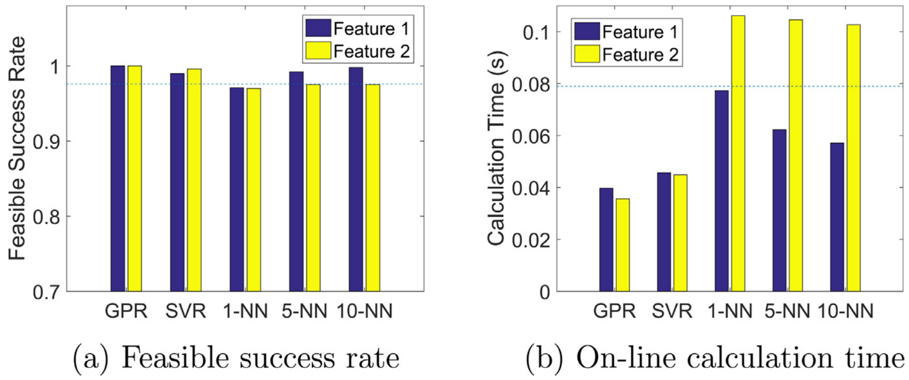

Fig. 7. Optimization results with different features.

built the optimal database D. Similarly, we randomly selected another 1000 groups of $\boldsymbol{X}$ in the working range to obtain a test dataset D_T. We took the data in D_T as the input and solved the optimization problems with Model 2 and the SQP solver. The initial guesses for the iterative solver were selected by the multivariable-multiple regression method with different feature selections and regression methods. Randomly selected initializations were used for comparison [16].

6.2.1. Feature selection for multivariable-multiple regression

To employ the multivariable-multiple regression method presented in Section 5.2, we selected the feature vector for each regression component in two ways. The first approach directly selected the input variable $\boldsymbol{X}$ as the feature vector for each component (Feature 1):

$$\boldsymbol{X_F}(i) = \boldsymbol{X} \ \ (i = 1, 2, \ldots, N_J + 1).$$

The second selected a specific feature vector for each component (Feature 2):

$$\boldsymbol{X_F}(i) = ({q_f}^{(i)}, {\omega_0}^{(i)}, t_f) \ \ (i = 1, 2, \ldots, N_J),$$

$$\boldsymbol{X_F}(N_J + 1) = \boldsymbol{X}.$$

Feature 1 is simple but may lead to lower efficiency of regression model building and lower accuracy of prediction, as the results are influenced by some less related variables. Feature 2 is constructed by considering the character of the specific case. As all the joints are required to reach the objective positions at the same time, the motion time is directly influenced by all the joints. Thus, the feature vector of the regression model for motion time t_f is the exact $\boldsymbol{X}$. As far as the regression of ω_m of each joint is concerned, only the components related to the corresponding joint and the joint motion time are relevant.

For Features 1 and 2, the GPR, SVR with a Gaussian kernel (SVR-G), 1-NN, 5-NN and 10-NN methods were adopted to select the initial guesses. The optimization results are shown in Fig. 7. The dashed line represents the result obtained with randomly selected initializations. With GPR and SVR-G, both Feature 1 and Feature 2 had better performance than arbitrarily initialized solutions. Feature 2 performed better than Feature 1, with a higher feasible success rate and shorter calculation time. With 5-NN and 10-NN, Feature 1 performed better than arbitrarily initialized solutions. Feature 2 did not perform well, as it required calculating the similarity and selecting the nearest neighbours for each component, which increased the calculation amount.

6.2.2. Regression methods and database size

Five regression methods were employed to select initializations: GPR, SVR-G, 1-NN, 5-NN, and 10-NN. Feature 2 was selected for GPR and SVR-G, and Feature 1 was selected for k-NN. We also extracted subsets in D with different sizes and evaluated the performance of the regression methods with different data amounts.

The performances of the regression methods with respect to the database size are displayed in Fig. 8. The dashed line represents the result obtained with randomly selected initializations. As shown in Fig. 8(a), the feasible success rate was improved by adopting the machine learning method. Initializing by GPR, SVR-G, 5-NN and 10-NN, the feasible success rates of the optimization solver were higher than those of the randomly initialized situation. Furthermore, using GPR and SVR-G, a high feasible success rate was achieved with a small number of samples, which approximated 100% with GPR when $N_D > 100$, or with SVR-G when $N_D > 500$.

As shown in Fig. 8(b), less on-line calculation time was achieved by adopting all five regression methods. 1-NN provided a slight improvement. The improvement increased as k increased. SVR-G performed even better. GPR was the best, reducing the on-line calculation time by more than 50%. The online calculation time first decreased and then increased as the database size decreased. The shortest on-line calculation time was obtained with $N_D = 5000$.

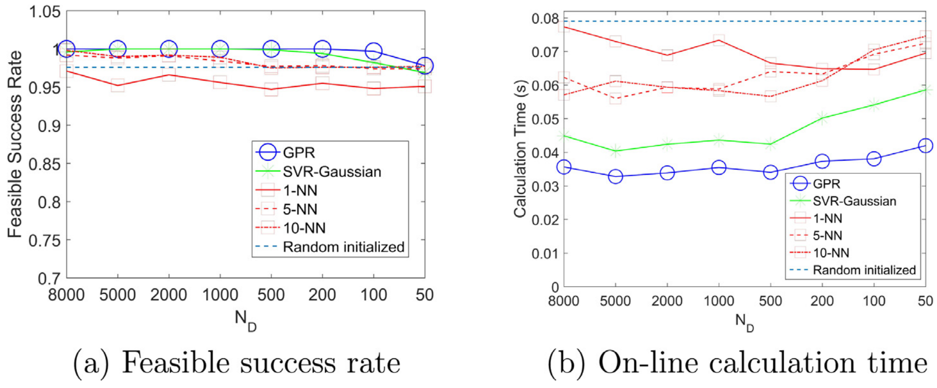

Fig. 8. Results with different regression methods and database sizes.

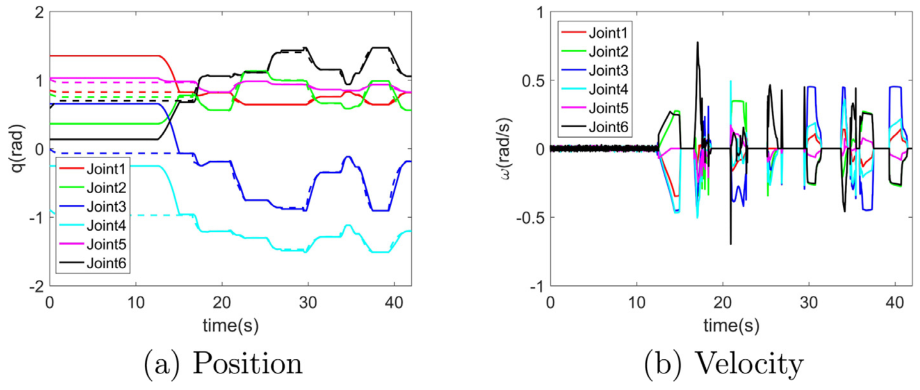

Fig. 9. Robot motion curves with a changing target.

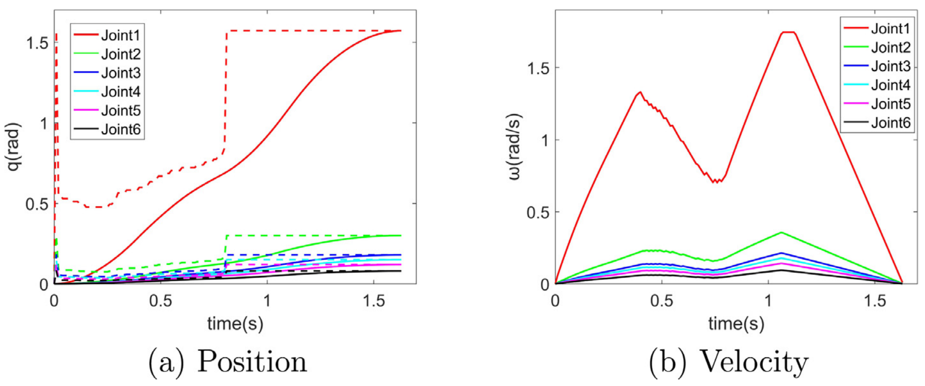

Fig. 10. Robot motion curves with moving obstacles.

6.3. Continuous trajectory planning in practical cases

To verify the practicality of the presented continuous trajectory planning method, we simulated the two practical cases described in Section 3.5.

We converted the algorithm into C code and ran it on a real-time system. The system consisted of an ABB IRB140 6-DOF robot [31,32], a Microsoft Kinect as the environment detection sensor, and a Linux real-time PC used to gather information from all the devices and calculate trajectories. The sampling time of the system was $T_p = 4$ms. We employed the 5-NN method with $N_D = 2000$ to perform machine-learning-based trajectory planning.

In the first case, we considered a human–robot interaction application. The robot aimed to move the end-effector to the position specified by the human hand to perform tasks such as hand-over or haptic interaction. The final goal configuration continuously changed according to the hand motion. The motion curves are shown in Fig. 9. The goal position and the planned position are represented by the dashed and solid lines, respectively. In this process, the robot reacted to the changing target quickly and tracked the target precisely.

In the second case, we considered a moving collision avoidance situation. The robot moved from the initial configuration to the goal configuration while a moving obstacle was passing the workspace. The motion curve is shown in Fig. 10. The goal position and the planned position for each planning cycle are represented by the dashed and solid lines, respectively. The final goal was $\boldsymbol{q_f} = [1.57, 0.30, 0.18, 0.15, 0.12, 0.08]$ (rad). From 0.020 s to 0.808 s, the temporal goal continued to proceed along the initially planned path according to the motion of the moving obstacle. At 0.808 s, the obstacle had passed the collision possible area, and the goal was set back to the final goal. In the whole process, the robot was slowed down in the middle to yield to the moving obstacle and finally reached the actual goal.

In the above situations, the average calculation time in each planning period was 0.37 ms. The maximum calculation time was 2.96 ms, which was below 4 ms. Therefore, real-time performance was satisfied.

7. Conclusions

An efficient kinematic continuous trajectory planning framework based on global non-convex optimization is presented in this paper. The computational accuracy and efficiency of global optimization are greatly improved by non-convex optimization model transformation and the multivariable-multiple-regression-based solver.

The methods are verified by multiple numerical experiments. With the transformed model, the iterative solver starting from different feasible initializations tends to converge to the same local solution, which indicates that we can obtain the accurate global solution with a single arbitrary initialization. Additionally, the feasibility and efficiency of the iterative solver are improved by selecting a good initialization with multivariable-multiple regression. In the best case, the feasible success rate is nearly 100%, and the calculation time is reduced by approximately 50%. Moreover, the methodology is experimentally verified by cases of continuous trajectory planning towards changing targets in a 250 Hz real-time system with a 6-DOF serial robot.

However, when we consider optimization problems with higher dimensions, such as the cases with higher dimensional DOF robots, more complex environments or more complex dynamics factors, the computational complexity increases greatly due to the high non-linearity and coupling. Real-time dynamic trajectory planning for robots with high-dimensional DOFs working in complex environments is still challenging. In the future, we would like to study better trajectory representation and more efficient numerical computation methods to further improve the computational efficiency to achieve on-line operation for more complex cases. Additionally, we would like to explore the integration of this trajectory planning framework with higher-level motion planning and lower-level control to adapt it to more practical situations and achieve better performance.

Acknowledgment

The authors would like to thank the editors and reviewers for their instructive comments that helped to improve the quality of the paper. The authors also would like to thank Prof. Federico Pecora (Örebro University) for his discussions about adapting the model in practical applications.

Appendix A. Optimization problem for Model 1

Substituting Eq. (13) into constraint (12), the latter is always satisfied.

Then, we substitute Eq. (13) into constraint (8). For the first inequality ($\omega_0{}^i \le \omega_m{}^i$), we obtain

$$\sqrt{(a^i t_f + \omega_0{}^i)^2 - 2(\omega_0{}^i)^2 - 4q_f{}^i a^i} \le a^i t_f - \omega_0{}^i, \tag{A.1}$$

which requires the three conditions below to hold at the same time:

$$(a^i t_f + \omega_0{}^i)^2 - 2(\omega_0{}^i)^2 - 4q_f{}^i a^i \ge 0, \tag{A.2}$$

$$a^i t_f - \omega_0{}^i \ge 0, \tag{A.3}$$

$$(a^i t_f + \omega_0{}^i)^2 - 2(\omega_0{}^i)^2 - 4q_f{}^i a^i \le (a^i t_f - \omega_0{}^i)^2, \tag{A.4}$$

Inequality (A.4) is derived by squaring both sides of inequality (A.1) and can be simplified as

$$\omega_0{}^i a^i t_f - q_f{}^i a^i - (\omega_0{}^i)^2/2 \le 0. \tag{A.5}$$

For the second inequality ($\omega_m{}^i \le \omega_{max}{}^i$), we obtain

$$a^i t_f + \omega_0{}^i - 2\omega_{max}{}^i \le \sqrt{(a^i t_f + \omega_0{}^i)^2 - 2(\omega_0{}^i)^2 - 4q_f{}^i a^i},$$

which requires one of the two conditions below to hold:

$$a^i t_f + \omega_0{}^i - 2\omega_{max}{}^i < 0, \tag{A.6}$$

$$a^i t_f + \omega_0{}^i - 2\omega_{max}{}^i \geq 0, \tag{A.7}$$

and

$$(a^i t_f + \omega_0{}^i - 2\omega_{max}{}^i)^2 \leq (a^i t_f + \omega_0{}^i)^2 - 2(\omega_0{}^i)^2 - 4q_f{}^i a^i. \tag{A.8}$$

We denote the feasible set specified by constraint (i) as Si. Then, the whole feasible set of this optimization problem can be described as

$$S7 \cap SA.2 \cap SA.3 \cap SA.5 \cap [SA.6 \cup (SA.7 \cap SA.8)]$$

Then, the optimization problem consists of finding parameters to minimize Eq. (6) in this feasible set.

By calculating the Hessian matrix and the leading principal minors, we can obtain that SA.3, SA.7 and SA.8 are convex, while SA.2, SA.5 and SA.6 are non-convex. Thus, this feasible set is non-convex and has a very complex geometric shape.

References

[1] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, P. Abbeel, Motion planning with sequential convex optimization and convex collision checking, Int. J. Robot. Res. 33 (9) (2014) 1251–1270.
[2] B. Bäuml, T. Wimböck, G. Hirzinger, Kinematically optimal catching a flying ball with a hand-arm-system, in: Proceedigns of the IROS, 2010, pp. 2592–2599.
[3] P.M. Pardalos, H.E. Romeijn, Handbook of Global Optimization, 2, Springer Science & Business Media, 2013.
[4] Z. Ren, Q. Zhu, R. Xiong, Trajectory planning of 7-dof humanoid manipulator under rapid and continuous reaction and obstacle avoidance environment, Acta Autom. Sin. 41 (6) (2015) 1131–1144.
[5] S.S.M. Salehian, M. Khoramshahi, A. Billard, A dynamical system approach for softly catching a flying object: theory and experiment, IEEE Trans. Robot. 32 (2) (2016) 462–471.
[6] J. Kober, M. Glisson, M. Mistry, Playing catch and juggling with a humanoid robot, in: Proceedigns of the Humanoids, 2012, pp. 875–881.
[7] J. Mainprice, R. Hayne, D. Berenson, Goal set inverse optimal control and iterative replanning for predicting human reaching motions in shared workspaces, IEEE Trans. Robot. 32 (4) (2016) 897–908.
[8] T. Iqbal, S. Rack, L.D. Riek, Movement coordination in human–robot teams: a dynamical systems approach, IEEE Trans. Robot. 32 (4) (2016) 909–919.
[9] A.M. Zanchettin, P. Rocco, Motion planning for robotic manipulators using robust constrained control, Control Eng. Pract. 59 (2) (2017) 127–136.
[10] S. Zhang, S. Dai, Real-time trajectory generation for haptic feedback manipulators in virtual cockpit systems, J. Comput. Inf. Sci. Eng. 18 (4) (2018) 041015.
[11] L. Biagiotti, C. Melchiorri, Trajectory Planning for Automatic Machines and Robots, Springer Berlin Heidelberg, 2009.
[12] H. Liu, X. Lai, W. Wu, Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints, Robot. Comput.-Integr. Manuf. 29 (2) (2013) 309–317.
[13] Z. Liu, J. Xu, Q. Cheng, Y. Zhao, Y. Pei, C. Yang, Trajectory planning with minimum synthesis error for industrial robots using screw theory, Int. J. Precis. Eng. Manuf. 19 (2) (2018) 183–193.
[14] T. Kröger, F.M. Wahl, Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events, IEEE Trans. Robot. 26 (1) (2010) 94–111.
[15] O. von Stryk, M. Schlemmer, Optimal control of the industrial robot Manutec r3, in: R. Bulirsch, D. Kraft (Eds.), Computational Optimal Control. ISNM International Series of Numerical Mathematics, vol. 115, Birkhäuser Basel, 1994.
[16] T. Chettibi, H. Lehtihet, M. Haddad, S. Hanchi, Minimum cost trajectory planning for industrial robots, Eur. J. Mech.-A/Solids 23 (4) (2004) 703–715.
[17] A.R. Kan, G.T. Timmer, Stochastic global optimization methods part i: clustering methods, Math. Programm. 39 (1) (1987) 27–56.
[18] A. Cassioli, D. Di Lorenzo, M. Locatelli, F. Schoen, M. Sciandrone, Machine learning for global optimization, Comput. Opt. Appl. 51 (1) (2012) 279–303.
[19] J. Pan, Z. Chen, P. Abbeel, Predicting initialization effectiveness for trajectory optimization, in: Proceeding of the ICRA, 2014, pp. 5183–5190.
[20] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, J. Peters, Trajectory planning for optimal robot catching in real-time, in: Proceedings of the ICRA, 2011, pp. 3719–3726.
[21] N. Jetchev, M. Toussaint, Fast motion planning from experience: trajectory prediction for speeding up movement generation, Autonom. Robots 34 (1) (2013) 111–127.
[22] K. Hauser, Learning the problem-optimum map: analysis and application to global optimization in robotics, IEEE Trans. Robot. 33 (1) (2017) 141–152.
[23] D. Tesar, M. Thomas, Dynamic modeling of serial manipulator arms, Trans. ASME 104 (1982) 218–228.
[24] O. Koç, G. Maeda, J. Peters, Online optimal trajectory generation for robot table tennis, Robot. Auton. Syst. 105 (2018) 121–137.
[25] S. Zhang, A.M. Zanchettin, R. Villa, S. Dai, Real-time trajectory planning based on joint-decoupled optimization in human-robot interaction, Mech. Mach. Theory 144 (2020) 103664.
[26] D. Sidobre, X. Broquère, J. Mainprice, E. Burattini, A. Finzi, S. Rossi, M. Staffa, Human–Robot Interaction, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 123–172.
[27] S. Pellegrinelli, A. Orlandini, N. Pedrocchi, A. Umbrico, T. Tolio, Motion planning and scheduling for human and industrial-robot collaboration, CIRP Ann. 66 (1) (2017) 1–4.
[28] J. Vannoy, J. Xiao, Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes, IEEE Trans. Robot. 24 (5) (2008) 1199–1212.
[29] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge university press, 2004.
[30] J.D. Kelleher, B. Mac Namee, A. D'arcy, Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies, MIT Press, 2015.
[31] R. Maderna, A. Casalino, A.M. Zanchettin, P. Rocco, Robotic handling of liquids with spilling avoidance: a constraint-based control approach, in: Proceedings of the ICRA, 2018, pp. 7414–7420.
[32] ABB, Irb140 industrial robot. data sheet, 2018. (http://new.abb.com/products/robotics/industrial-robots/irb-140/irb-140-data).