

Article

Smooth and Time-Optimal Trajectory Planning for Robots Using Improved Carnivorous Plant Algorithm

Bo Wei ^{1,*}, Changyi Liu ¹, Xin Zhang ¹, Kai Zheng ¹, Zhengfeng Cao ¹ and Zexin Chen ²

¹ School of Advanced Manufacturing Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China; s222101011@stu.cqupt.edu.cn (C.L.); s212101030@stu.cqupt.edu.cn (X.Z.)

² Menlo Intelligence Company, Chongqing 518129, China; 13603003817@163.com

* Correspondence: bowei@cqupt.edu.cn

Abstract: To improve the safety and reliability of robotic manipulators during high-speed precision movements, this paper proposes a method for smooth and time-optimal trajectory planning incorporating kinodynamic constraints. The primary objective is to use an evolutionary algorithm to determine a trajectory by considering time and jerk within the feasible path-pseudo-velocity phase plane region. Firstly, the path parameterization theory extracted the maximum pseudo-velocity projection curve from the kinodynamic constraints. Subsequently, the feasible region in the phase plane was defined through reachability analysis of discrete linear systems. Thereafter, we constructed the trajectory function using a cubic B-spline curve, optimizing its control points with an improved carnivorous plant optimization algorithm. Finally, the effectiveness and practicality of this method were verified through simulations on a 6-DOF manipulator.

Keywords: trajectory planning; jerk; improved carnivorous plant algorithm; trajectory optimization



Citation: Wei, B.; Liu, C.; Zhang, X.; Zheng, K.; Cao, Z.; Chen, Z. Smooth and Time-Optimal Trajectory Planning for Robots Using Improved Carnivorous Plant Algorithm.

Machines **2024**, *12*, 802. <https://doi.org/10.3390/machines12110802>

Academic Editor: Dan Zhang

Received: 17 October 2024

Revised: 8 November 2024

Accepted: 9 November 2024

Published: 12 November 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Introducing dynamic constraints is crucial for improving the safety and reliability of mobile robots, especially during high-speed manipulator operations. Limiting jerk helps reduce mechanical strain, avoid resonance, and prevent actuator damage, ensuring smoother, more precise trajectories. While third-order constraints (jerk-based) enhance trajectory smoothness and component durability, they come with higher computational costs. Second-order constraints (acceleration-based) are sufficient for speed-focused tasks, but third-order constraints are essential for applications prioritizing smooth motion and longevity. In summary, smooth, time-optimal trajectory planning with third-order dynamic constraints improves system performance and ensures accurate, efficient operation.

One solution of trajectory planning is to interpolate the position and time series of joints with smooth curves at several intermediate points of the geometric path, and then optimize the specific coefficient of the interpolation function according to the relevant constraints and performance indexes. In earlier studies, Lin et al. used cubic interpolation from splines to approximate the path and optimized the spline node position by the local iterative polyhedral search method [1]. Gasparetto and Zanotto [2] proposed quintic B-splines in joint trajectory interpolation and optimized the trajectory using sequential quadratic programming to further ensure the continuity of the trajectory. In 2008, they proposed an optimal trajectory planning technology for a robot manipulator. In order to obtain the optimal trajectory, Gasparetto and Zanotto minimized the objective function, consisting of the total execution time and the square integral of the jerk (the derivative of acceleration) [3]. Wang et al. employed a quintic B-spline function to interpolate the trajectory and combined it with a hybrid WOA-GA algorithm to realize the time-jerk optimal trajectory under kinematic constraints [4]. Machmudah et al. successfully planned the joint motion trajectory with the optimal time and torque by using the high-order polynomial curve method, combined with the genetic algorithm and particle swarm

optimization algorithm. The kinematic and dynamic constraints occur at the same time in the environment where the manipulators are facing obstacles [5]. However, trajectory planning directly in joint space is a complex, high-dimensional problem. While existing methods are capable of generating trajectories that meet optimal performance metrics, they rely on a large number of parameters and high-order interpolation operations. This dependence increases the calculation's time complexity and may reduce motion accuracy due to fluctuations in the interpolation function's coefficient.

Another solution involves transforming the many-body differential equations and system constraints into one-dimensional motion along a specific geometric path, reducing the multi-dimensional state space of a robotic system to a low-dimensional state space [6]. In this context, Bobrow et al. [7], along with Shin and McKay [8], pioneered the idea of characterizing the equations of motion using scalar path parameters and their time derivatives. They parameterized the joint torque constraints to path-curve coordinates, making it possible to find a time-optimal curve in the path-pseudo-velocity phase plane. Building on this, Shiller and Lu [9] extended this work by transforming the original problem into an equivalent reduction problem, replacing the torque constraint with a constraint on maximum pseudo-acceleration and pseudo-velocity along the path. Verschuere et al. converted the time-optimal trajectory planning (TOTP) problem into a convex optimization problem [10]. They expressed the original problem as a large sparse optimization problem after discretization, then adopted second-order cone programming to improve solving efficiency. To avoid abrupt changes in torque, Oberherber et al. used spline curves to approximate the planning results in the phase plane, optimizing the control points of the spline curves [11]. Pham et al. proposed a Reachability Analysis-based TOPP method (TOPP-RA) [12], focusing on solving the time optimization problem under second-order constraints. By carefully controlling the bidirectional numerical integration process and using the dichotomous algorithm to address constraint violations during integration, E. Barnett et al. proposed the widely applicable bisection algorithm for TOTP (TOTP-BA) method [13]. Lin presented a unified methodology to find the near optimal solution of a minimum-jerk trajectory based on PSO [14]. Kucuk [15] combined cubic spline with the seventh order polynomial to construct minimum-time smooth motion trajectories beginning and ending with zero jerk using a PSO algorithm for serial and parallel manipulators. Huang et al. [16] also described a methodology adopting the quintic B-spline interpolation in joint space but applied a non-dominated sorting genetic algorithm (NSGA-II) as a multi-objective optimization algorithm to obtain a time-jerk optimal trajectory.

Despite significant progress in these parameterization solutions, satisfying third-order constraints and finding optimal trajectories in multi-objective environments remains challenging. Faced with these challenges, we draw inspiration from previous discussions by analyzing the convex polygonal region formed by the first to third order constraints of discrete path points within the path-pseudo-velocity phase plane, allowing us to extract the maximum pseudo-velocity curve. Additionally, we further define the feasible region by assessing the reachability of adjacent path points. To reduce the dimensionality of the variables during the optimization process, we selected the control points of the cubic B-spline curve as our optimization variables and employed an improved carnivorous plant optimization algorithm for solving. Throughout the iterative resolution process, we took into account both efficiency and smoothness to ensure an optimal outcome.

The rest of this paper is organized as follows. The constraints in trajectory planning and the parameterization of the smooth and time-optimal path are discussed in the upcoming section. Then, the reconstruction of feasible regions and the mathematical model for applying cubic B-spline interpolation in the phase plane are presented in Section 3. In Section 4, the Carnivorous Plant Algorithm and its enhancements are introduced. Section 5 provides an analysis of the numerical simulation results. Finally, Section 6 summarizes the key findings and conclusions of this study.

2. Modeling Optimization Problems

2.1. Path Parameterization

Consider a smooth path represented in the joint space coordinates, denoted as $\theta(s)$, where s represents a scalar path parameter. The parameter s describes the geometric shape of the path, and its mapping relationship with time t , $s(t) : \mathbb{R} \rightarrow [0, \Lambda]$, reveals the law of trajectory change over time. Furthermore, to ensure continuous advancement of the robot along the path of total length Λ , it must satisfy $\dot{s}(t) \geq 0$.

The joint velocity, acceleration, and jerk of the robot can be determined by the chain derivation rule as follows:

$$\begin{cases} \dot{q} = \dot{s}q' \\ \ddot{q} = \ddot{s}q' + \dot{s}^2 q'' \\ \dddot{q} = \dddot{s}q' + 3\ddot{s}\dot{s}q'' + \dot{s}^3 q''' \end{cases} \quad (1)$$

In this equation, $q' = \partial q(s)/\partial s$, $q'' = \partial^2 q(s)/\partial s^2$, and $q''' = \partial^3 q(s)/\partial s^3$ denote the first, second, and third derivatives of the joint position with respect to s . Similarly, $\dot{s} = ds/dt$, $\ddot{s} = d^2s/dt^2$ and $\dddot{s} = d^3s/dt^3$ represent the pseudo-velocity, pseudo-acceleration, and pseudo-jerk, respectively.

For an n -axis robot with rigid components, the complete dynamic equation under a nonlinear friction model can be expressed as follows:

$$\tau(q, \dot{q}, \ddot{q}) = M(q)\ddot{q} + C(q)\dot{q} + G(q) \quad (2)$$

where $\tau \in \mathbb{R}^n$ is the joint force vector; $M(q) \in \mathbb{R}^{n \times n}$ is the inertia matrix; $C(q) \in \mathbb{R}^{n \times n}$ is the Coriolis and centripetal force matrix; and $G(q) \in \mathbb{R}^n$ is the gravity term vector.

We substitute Equation (1) into Equation (2) to achieve the mapping of joint torques to s space:

$$\begin{cases} \tau(s, \dot{s}, \ddot{s}) = m(s)\ddot{s} + c(s)\dot{s}^2 + g(s) \\ m(s) = M(q)q' \\ c(s) = M(q)q'' + q'^T C(q)q' \\ g(s) = G(q) \end{cases} \quad (3)$$

To ensure smooth and precise movement of the robot during trajectory movement, it is necessary to control the rate of torque. Calculate the time derivative from Equation (3):

$$\begin{cases} \dot{\tau}(s, \dot{s}, \ddot{s}, \dddot{s}) = m(s)\dddot{s} + p(s)\ddot{s}\dot{s} + c'(s)\dot{s}^3 + g'(s)\dot{s} \\ p(s) = m(s) + 2c(s) \end{cases} \quad (4)$$

2.2. Objective Function Parameterization

The objective function combines task completion time and normalized jerk (the derivative of acceleration). For a robot moving along a predetermined path, the total travel time is defined as follows:

$$\int_0^T 1dt = \int_0^\Lambda \left(\frac{ds}{dt}\right)^{-1} ds = \int_0^\Lambda \frac{1}{\dot{s}} ds \quad (5)$$

The smoothness of the trajectory is assessed by integrating the squared normalized jerk of the joints:

$$\int_0^T \lambda \sum_k^n \frac{(\ddot{q}_k)^2}{(\overline{\ddot{q}}_k)} dt = \int_0^\Lambda \lambda \sum_k^n \frac{(\ddot{q}_k)^2}{(\overline{\ddot{q}}_k)} \left(\frac{ds}{dt}\right)^{-1} ds = \int_0^\Lambda \lambda \sum_k^n \frac{(\ddot{q}_k)^2}{(\overline{\ddot{q}}_k)} \frac{1}{\dot{s}} ds \quad (6)$$

where λ is a weighting factor used to balance different factors, and for $k = 1 \dots n$, $\overline{\ddot{q}}_k = \max_{s \in [0, \Lambda]} \ddot{q}_k(s)$.

Based on this, the smooth time-optimal trajectory planning problem can be formulated as the following constrained optimization problem:

$$\begin{aligned} \min_{s(t)} J &= \int_0^\Lambda \frac{1}{\dot{s}} + \lambda \sum_k^n \frac{(\ddot{q}_k)^2}{(\ddot{q}_k)} \frac{1}{\dot{s}} ds \\ \text{s.t.} &\begin{cases} \tau_{\min} \leq m(s)\ddot{s} + c(s)\dot{s}^2 + g(s) \leq \tau_{\max} \\ \dot{\tau}_{\min} \leq m(s)\ddot{s} + p(s)\ddot{s}\dot{s} + c'(s)\dot{s}^3 + g'(s)\dot{s} \leq \dot{\tau}_{\max} \\ \dot{q}_{\min} \leq q'(s)\dot{s} \leq \dot{q}_{\max} \\ \ddot{q}_{\min} \leq q'(s)\ddot{s} + q''(s)\dot{s}^2 \leq \ddot{q}_{\max} \\ \ddot{q}_{\min} \leq q'(s)\ddot{s} + 3q''(s)\dot{s}\ddot{s} + q'''(s)\dot{s}^3 \leq \ddot{q}_{\max} \\ s(0) = 0, s(T) = \Lambda, \dot{s}(0) = \dot{s}_0, \dot{s}(T) = \dot{s}_T, \\ \dot{s}(t) \geq 0, t \in [0, T] \end{cases} \end{aligned} \quad (7)$$

3. Optimization Problem Solving

3.1. Feasibility Region Analysis in the Phase Plane

The optimization variable $s(t)$ constitutes a typical functional optimization problem. Directly solving this problem is challenging; hence, we simplify it into a finite discrete form.

Taking Δs as the step size, we sample along the path length, dividing the interval $[0, \Lambda]$ into N segments and $N + 1$ grid points $0 =: s_0, s_1 \dots s_{N-1}, s_N := \Lambda$. For each grid point s , we identify a series of surface regions with varying shapes. We arrange these surface regions in order to form a four-dimensional state space $(s, \dot{s}, \ddot{s}, \ddot{s})$ defined by the boundaries of these surfaces. Within this high-dimensional space, motion is constrained independently of the initial conditions. When mapping from the subspace of $(\dot{s}, \ddot{s}, \ddot{s})$ to the subspace of $(\dot{s}^2, \ddot{s})$, we observe that the set of linear inequalities formed by second-order constraints delineates a convex polygonal region, as shown in Figure 1.

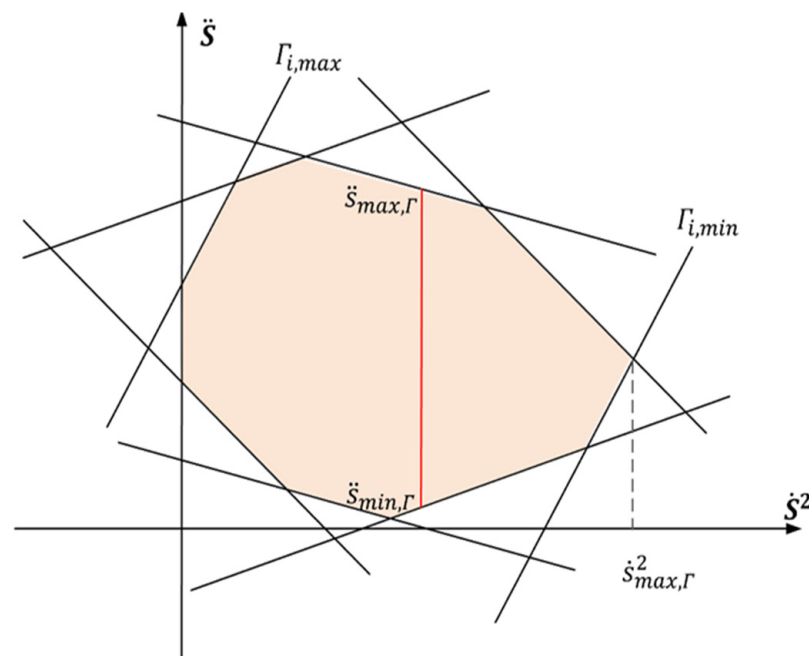


Figure 1. Convex polygonal region on the $\dot{s}^2 - \ddot{s}$ plane.

These constraints can be expressed by the following inequalities:

$$\Gamma_{i,\min} \leq A_i \dot{s} + B_i \dot{s}^2 + C_i \leq \Gamma_{i,\max}, i = 1, \dots, 2n, \quad (8)$$

where $A_i, B_i, C_i, \Gamma_{i,min}$ and $\Gamma_{i,max}$ represent the elements in the vectors of physical quantities under the kinematic constraints $\ddot{q}$ and dynamic constraints τ in model (7), respectively.

For each value of s , the pseudo-velocity and pseudo-acceleration are limited by the following:

$$\begin{aligned} 0 \leq \dot{s} \leq \dot{s}_{max,\Gamma}(s) \\ \ddot{s}_{min,\Gamma}(s, \dot{s}) \leq \ddot{s} \leq \ddot{s}_{max,\Gamma}(s, \dot{s}) \end{aligned} \quad (9)$$

where:

$$\begin{aligned} \dot{s}_{max,\Gamma}(s) &= \min_{i,j} \left\{ \max_{\Gamma_i, \Gamma_j} \left(\frac{A_j(\Gamma_i - C_i) - A_i(\Gamma_j - C_j)}{A_j B_i - A_i B_j} \right) \right\}, i, j = 1, \dots, 2n, \\ \ddot{s}_{min,\Gamma}(s, \dot{s}) &= \max_i \left\{ \min_{\Gamma_i} \left(\frac{\Gamma_i - C_i - B_i \dot{s}^2}{A_i} \right) \right\}, i = 1, \dots, 2n, \\ \ddot{s}_{max,\Gamma}(s, \dot{s}) &= \min_i \left\{ \max_{\Gamma_i} \left(\frac{\Gamma_i - C_i - B_i \dot{s}^2}{A_i} \right) \right\}, i = 1, \dots, 2n, \end{aligned} \quad (10)$$

Similarly, when mapping to the $(\ddot{s}, \ddot{\ddot{s}})$ subspace, a set of linear inequalities formed by third-order constraints also defines another convex polygonal region, as shown in Figure 2. These third-order constraints are expressed as follows:

$$\Psi_{i,min} \leq a_i \ddot{\ddot{s}} + b_i^* \ddot{s} + c_i^* \leq \Psi_{i,max}, i = 1, \dots, 2n, \quad (11)$$

where $b_i^* = b_i \dot{s}, c_i^* = c_i \dot{s}^3 + h_i \dot{s}; a_i, b_i, c_i, \Psi_{i,min}$ and $\Psi_{i,max}$ represent the elements in the vectors of physical quantities under the kinematic constraints $\ddot{q}$ and dynamic constraints $\dot{\tau}$ in model (7), respectively.

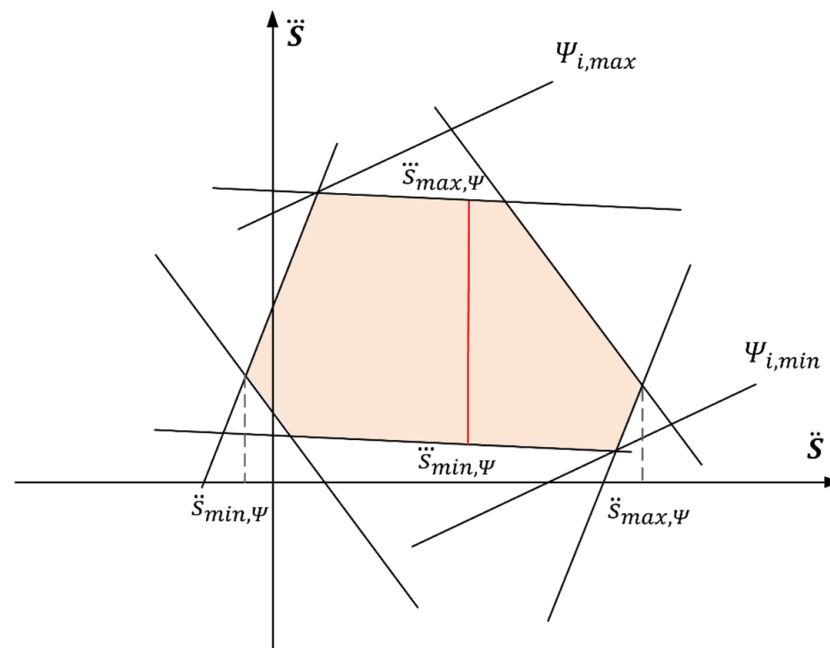


Figure 2. Convex polygonal region on the $\ddot{s} - \ddot{\ddot{s}}$ plane.

For each set of values for s and $\dot{s}$, the boundaries for the pseudo-acceleration $\ddot{s}$ and the pseudo-jerk $\ddot{\ddot{s}}$ induced by the third-order constraints are as follows:

$$\begin{aligned} \ddot{s}_{min,\Psi}(s, \dot{s}) \leq \ddot{s} \leq \ddot{s}_{max,\Psi}(s, \dot{s}) \\ \ddot{\ddot{s}}_{min}(s, \dot{s}, \ddot{s}) \leq \ddot{\ddot{s}} \leq \ddot{\ddot{s}}_{max}(s, \dot{s}, \ddot{s}) \end{aligned} \quad (12)$$

where:

$$\begin{aligned}\ddot{s}_{min,\Psi}(s, \dot{s}) &= \max_{i,j} \left\{ \min_{\Psi_i, \Psi_j} \left(\frac{a_j(\Psi_i - c_i^*) - a_i(\Psi_j - c_j^*)}{a_j b_i^* - a_i b_j^*} \right) \right\}, i, j = 1, \dots, 2n \\ \ddot{s}_{max,\Psi}(s, \dot{s}) &= \min_{i,j} \left\{ \max_{\Psi_i, \Psi_j} \left(\frac{a_j(\Psi_i - c_i^*) - a_i(\Psi_j - c_j^*)}{a_j b_i^* - a_i b_j^*} \right) \right\}, i, j = 1, \dots, 2n \\ \ddot{s}_{min}(s, \dot{s}, \ddot{s}) &= \max_i \left\{ \min_{\Psi_i} \left(\frac{\Psi_i - c_i^* - b_i^* \ddot{s}}{a_i} \right) \right\}, i = 1, \dots, 2n \\ \ddot{s}_{max}(s, \dot{s}, \ddot{s}) &= \min_i \left\{ \max_{\Psi_i} \left(\frac{\Psi_i - c_i^* - b_i^* \ddot{s}}{a_i} \right) \right\}, i = 1, \dots, 2n\end{aligned}\quad (13)$$

We can redefine the first-order constraint $\dot{q}$ in model (7) as $|\dot{s}_{max,q} q'| \leq \dot{q}$. For a fixed grid point s , set $\dot{s}$ as $\min(\dot{s}_{max,q}(s), \dot{s}_{max,\Gamma}(s))$. Then, using a vector positioning algorithm, quickly determine whether the space enclosed by the third-order constraints within the $\dot{s} - \ddot{s}$ plane is non-empty.

If not, calculate the pseudo-acceleration range $[\ddot{s}_{min,\Psi}(s, \dot{s}), \ddot{s}_{max,\Psi}(s, \dot{s})]$ according to Equation (13) and verify whether this range intersects with the reference range $[\ddot{s}_{min,\Gamma}(s, \dot{s}), \ddot{s}_{max,\Gamma}(s, \dot{s})]$ defined by Equation (11). If the initial space is empty or there is no intersection in the pseudo-acceleration range, gradually reduce $\dot{s}$ until an intersection is found.

By implementing the above method at each grid point on the path, it becomes possible to construct a set S_{sd} of $(s, \dot{s})$ that meets all the constraints, obtaining the projection of the high-dimensional state space boundary onto the $s - \dot{s}$ plane.

Since calculations in high-dimensional space are complex and time-consuming, simplifying the problem to the $s - \dot{s}$ plane can improve algorithm efficiency and speed. This “maximum pseudo-velocity projection” reveals the limited relationship between the path and joint speeds and narrows down the feasible region on the $s - \dot{s}$ plane.

Furthermore, by employing Model Predictive Control reachability theory, we can analyze the constraint relationships between path points, and we can further refine the feasible region range on the $s - \dot{s}$ plane. Given that $\frac{d\dot{s}^2}{ds} = 2\ddot{s}$, the velocity relationship between adjacent path points can be discretized as follows:

$$\begin{aligned}x_{i+1} &= x_i + 2\Delta u_i, i = 0 \dots N - 1 \\ x_i &\in \mathcal{X}_i, u_i \in \mathcal{U}_i\end{aligned}\quad (14)$$

where x_i is the state variable at stage i , representing the squared pseudo-velocity $\dot{s}_i^2$ at point s_i ; u_i is the control variable at stage i , representing a constant path pseudo-acceleration $\ddot{s}_i$ (covering the interval $[s_i, s_{i+1}]$); and $\mathcal{X}_i$ and $\mathcal{U}_i$ are the explicit constraints for the state variable and control variable at stage i , respectively.

The maximum pseudo-velocity projection curve obtained is defined as an explicit constraint for the state variable, and the second-order kinodynamic constraints are defined as an explicit constraint for the control variable. The control-state set Ω_i , defined jointly by these constraints, still maintains a convex polygonal shape:

$$\Omega_i := \{(u, x) | A_i u_i + B_i x_i + C_i \in \Gamma_i, \sqrt{x_i} \in S_{sd,i}\} \quad (15)$$

Equations (14) and (15) describe a discrete-time linear system constrained by linear control-state inequality conditions. Based on this system, the optimal parameterization is computed in two passes, as depicted in Figure 3. In the first pass, starting from the initial pseudo-velocity state along the predetermined path, the boundaries of the reachable set $\mathcal{L}_i$ are calculated stage by stage using a forward (fw) recursive method. The calculation process is as follows:

$$\begin{aligned}x^+ &:= \max_{(u, \tilde{x}) \in \Omega_i, x^+ \in \mathcal{X}_{i+1}} (\tilde{x} + 2\Delta u) \\ x^- &:= \min_{(u, \tilde{x}) \in \Omega_i, x^- \in \mathcal{X}_{i+1}} (\tilde{x} + 2\Delta u) \\ \mathcal{L}_i &:= \{x | x^- \leq x \leq x^+\}\end{aligned}\quad (16)$$

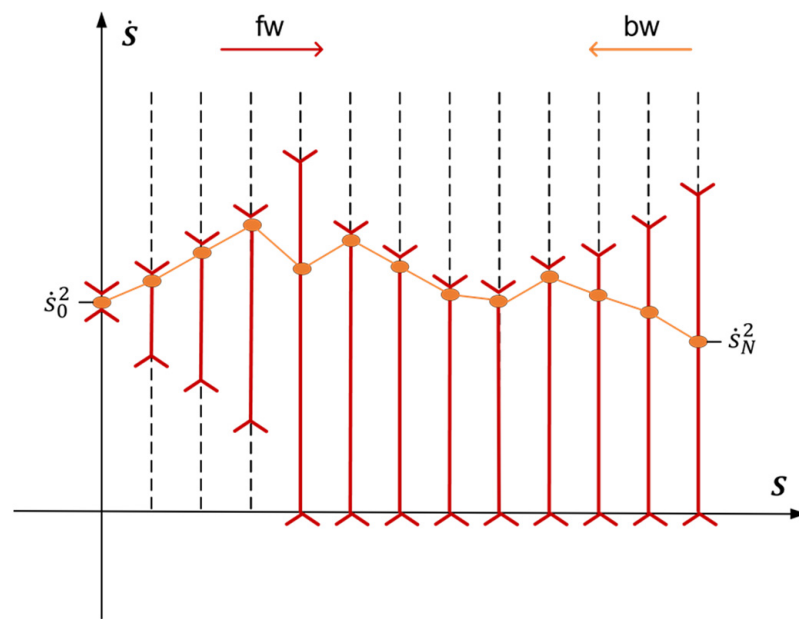


Figure 3. Determining the upper bound of the feasible region through reachability analysis.

In the second pass, starting from the path's end state and using a backward (bw) recursive minimum control strategy to ensure each state falls within the reachable set of its previous stage, the specific calculation formula is the following:

$$x_i^* := \min \{ x_{i+1}^* - 2\Delta \min(u_i), x_i^+ \} \quad (17)$$

Through this method, the backward phase trajectory x^* updates the upper bound of the feasible region.

3.2. Trajectory Construction

When constructing a discrete model to ensure that the pseudo-velocity curve meets the constraint conditions throughout the entire motion process, it is typically necessary to use densely sampled points. However, this approach increases the dimensionality of the optimization variables, thereby reducing the efficiency of model solving. To address this issue, we used cubic B-spline curves within the feasible region of the phase plane to construct third-order continuous (C^3) [17] pseudo-velocity curves, as shown in Figure 4.

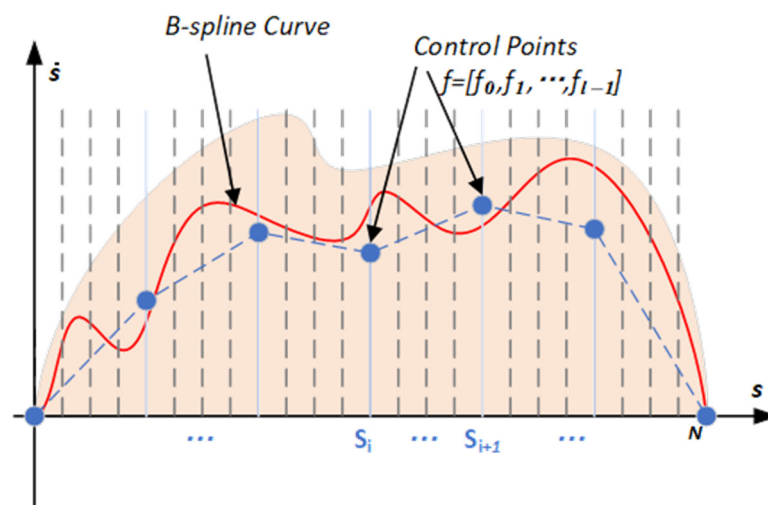


Figure 4. B-spline curves in the feasible region.

In the process of defining the B-spline curves, we selected a set of path nodes, denoted as $S = [S_0, S_1, \dots, S_{l-1}]$, where each pair of adjacent nodes encompasses mmm original path segments. Based on these nodes, we defined the corresponding pseudo-velocity vector $f = [f_0, f_1, \dots, f_{l-1}]$, where each node S_i corresponds to a pseudo-velocity vector component f_i . These components f_i , serving as control points, have a decisive impact on the geometric shape of the B-spline curve.

By substituting the optimization variable $s(t)$ in model (7) with the pseudo-velocity vector f , the discrete model for the optimal trajectory planning problem can be reformulated as follows:

$$\min_f J = \sum_{i=1}^N \frac{1}{\dot{s}(s_i)} \left(1 + \lambda \sum_k^n \frac{(\ddot{q}_{k,i})^2}{(\ddot{q}_{k,i})} \right) \Delta s \quad \text{s.t} \quad \begin{cases} \tau_{\min} \leq \tau(s_i, \dot{s}(s_i), \ddot{s}(s_i)) \leq \tau_{\max} \\ \dot{\tau}_{\min} \leq \dot{\tau}(s_i, \dot{s}(s_i), \ddot{s}(s_i), \ddot{s}(s_i)) \leq \dot{\tau}_{\max} \\ \dot{q}_{\min} \leq \dot{q}(s_i, \dot{s}(s_i)) \leq \dot{q}_{\max} \\ \ddot{q}_{\min} \leq \ddot{q}(s_i, \dot{s}(s_i), \ddot{s}(s_i)) \leq \ddot{q}_{\max} \\ \ddot{q}_{\min} \leq \ddot{q}(s_i, \dot{s}(s_i), \ddot{s}(s_i), \ddot{s}(s_i)) \leq \ddot{q}_{\max} \\ 0 \leq \dot{s}(s_i) \leq \sqrt{x_i^*} \end{cases} \quad (18)$$

Since the number of control points is far less than the number of sampling points, the dimensionality of the optimization variables is accordingly reduced, thereby speeding up the model solving process.

Despite the control points being relatively sparse, B-splines demonstrate a robust ability for shape expression, allowing the pseudo-velocity curve to be finely adjusted to complex shapes to fulfill model requirements. Furthermore, B-splines exhibit local support characteristics, meaning the interaction between distant control points is minimal. This feature significantly facilitates the rapid convergence of the pseudo-velocity curve during the optimization process. The pseudo-velocity curve is represented in cubic B-spline form as follows:

$$\dot{s}(u) = \sum_{i=0}^{n-1} N_{i,3}(u) f_i \quad (19)$$

Here, $u = \frac{s}{\Lambda}$ is a normalized path parameter, which maps the path length s to intervals $[0, 1]$.

In Equation (19), the basis function $N_{i,3}(u)$ in the cubic B-spline is calculated according to the following recursive relation:

$$N_{i,0}(u) = \begin{cases} 1, & \text{if } u_i \leq u \leq u_{i+1}, \\ 0, & \text{otherwise.} \end{cases} \quad N_{i,k}(u) = \frac{u-u_i}{u_{i+k}-u_i} N_{i,k-1}(u) + \frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}} N_{i+1,k-1}(u) \quad , k = 1, 2, 3. \quad (20)$$

The first and second derivatives of B-spline with respect to the normalized path parameter u are

$$\begin{aligned} \frac{d\dot{s}(u)}{du} &= \frac{dN_{0,3}(u)}{du} f_0 + \dots + \frac{dN_{n-1,3}(u)}{du} f_{n-1} = \sum_{i=0}^{n-1} \left(\frac{dN_{i,3}(u)}{du} f_i \right) \\ \frac{d^2\dot{s}(u)}{du^2} &= \frac{d^2N_{0,3}(u)}{du^2} f_0 + \dots + \frac{d^2N_{n-1,3}(u)}{du^2} f_{n-1} = \sum_{i=0}^{n-1} \left(\frac{d^2N_{i,3}(u)}{du^2} f_i \right) \end{aligned} \quad (21)$$

where

$$\frac{d^k N_{i,3}(u)}{du^k} = 3 \left(\frac{d^{k-1} N_{i,2}(u)}{du^{k-1}} \frac{1}{u_{i+3} - u_i} - \frac{d^{k-1} N_{i+1,2}(u)}{du^{k-1}} \frac{1}{u_{i+4} - u_{i+1}} \right), k = 1, 2$$

The parameter vector $\mathbf{u}$ used in the B-spline contains $n - 2$ uniformly distributed knots, calculated as follows:

$$\mathbf{u} = [0 \quad u_1 \quad u_2 \quad \cdots \quad 1] = \underbrace{[0 \quad \Delta s \quad 2\Delta s \quad \cdots \quad \Lambda]}_{n-2} \times \frac{1}{\Lambda} \quad (22)$$

where $\Delta s = \frac{\Lambda}{n-3}$.

To ensure that the B-spline passes through the starting and ending points, i.e., $s(0) = 0$, $f_0 = \dot{s}_0$ and $s(T) = \Lambda$, $f_{n-1} = \dot{s}_\Lambda$, the first and last elements of the parameter vector $\mathbf{u}$ are each repeated four times:

$$\mathbf{u} = \left[\underbrace{0 \cdots 0}_4 \quad u_i \quad u_{i+1} \quad \cdots \quad \underbrace{1 \cdots 1}_4 \right]. \quad (23)$$

Therefore, the calculations for the pseudo-acceleration $\ddot{s}$ and the pseudo-jerk $\dddot{s}$ along the path s are as follows:

$$\begin{aligned} \ddot{s}(s) &= \left(\frac{1}{\Lambda} \right) \left(\dot{s}(u) \frac{d\dot{s}(u)}{du} \right) \Big|_{u=\frac{s}{\Lambda}} \\ \dddot{s}(s) &= \left(\frac{1}{\Lambda} \right)^2 \left(\dot{s}(u)^2 \frac{d^2\dot{s}(u)}{du^2} \right) \Big|_{u=\frac{s}{\Lambda}} + \left(\left(\frac{1}{\Lambda} \dot{s}(u) \frac{d\dot{s}(u)}{du} \right) \Big|_{u=\frac{s}{\Lambda}} \right)^2 \end{aligned} \quad (24)$$

4. Improved Carnivorous Plant Algorithm

To address complex kinematic and dynamic constraints and optimize robot manipulation paths in multi-objective environments, previous studies have utilized heuristic algorithms to identify optimal trajectories that satisfy third-order constraints [14–17]. Building on these findings, we conducted an in-depth analysis and developed further enhancements for the cubic B-spline curve. Specifically, this paper introduces the Improved Carnivorous Plant Algorithm (ICPA) to optimize the distribution of control points in the cubic B-spline, meeting third-order constraint requirements while enhancing path smoothness and efficiency.

4.1. Carnivorous Plant Algorithm

The Carnivorous Plant Algorithm (CPA) [18], introduced by Ong in 2021, is a bio-inspired meta-heuristic designed to mimic the survival strategies and adaptation mechanisms of carnivorous plants in challenging environments. The algorithm demonstrates high performance in tackling complex search space problems characterized by high-dimensional design variables, multiple constraints, and numerous local optima. The CPA operates through four main stages: initialization, classification and grouping, growth, and reproduction.

During the initialization stage, CPA constructs an initial population of N individuals consisting of carnivorous plants (n_{CPlant}) and prey (n_{Prey}) through random allocation. Importantly, n_{Prey} should be k times of n_{CPlant} . The spatial distribution of these individuals is represented in matrix form.

$$Pop = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,D} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ X_{N,1} & X_{N,2} & \cdots & X_{N,D} \end{bmatrix} \quad (25)$$

where D represents the spatial dimension, and the sum of n_{CPlant} and n_{Prey} equals N . The random initialization of the population follows the equation:

$$X_{i,j} = L_{b_j} + (U_{b_j} - L_{b_j}) \times rand \quad (26)$$

Let $i = 1, 2, \dots, N$ and $j = 1, 2, \dots, D$, where L_b and U_b represent the lower and upper bounds of the search space, respectively. The term “*rand*” refers to a random number within the interval $[0, 1]$.

Entering the classification and grouping stage, the algorithm sorts each individual in descending order according to the fitness value, and then it labels the top n_{CPlant} individuals as carnivorous plants and the remaining n_{Prey} individuals as prey. The most suitable prey is assigned to the top carnivorous plant. Immediately afterward, the second and third most suitable prey are paired with the corresponding carnivorous plants. This process repeats until each carnivorous plant is paired with its corresponding prey. After that, the extra prey is paired again starting with the highest-ranking carnivorous plant, and the pattern continues. Take Figure 5 as an example: it shows the visualization of carnivorous plants and prey, in which the population size n is set to 6 and the coefficient k is 2.

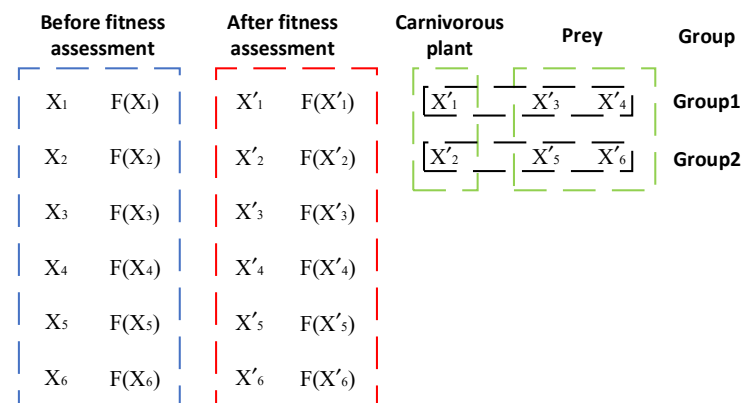


Figure 5. Grouping process with population size 6 and proportionality factor 2.

During the growth stage, which is part of the exploration phase, carnivorous plants release scents to attract prey. Captured prey provide nutrients that foster plant growth; however, not all preys are successfully captured. To account for this, an attraction rate (*ar*) is introduced, determining prey capture by comparing the *ar* value with a random number between 0 and 1. The equations used in this phase are as follows:

$$\begin{aligned} NewCP_{i,j} &= growth \times CP_{i,j} + (1 - growth) \times Prey_{v,j} \\ growth &= growth_{rate} \times rand_{i,j} \end{aligned} \quad (27)$$

$$\begin{aligned} NewPrey_{i,j} &= growth \times Prey_{u,j} + (1 - growth) \times Prey_{v,j}, \quad u \neq v \\ growth &= \begin{cases} growth_{rate} \times rand_{i,j} & \text{iff}(Prey_v) > f(Prey_u) \\ 1 - growth_{rate} \times rand_{i,j} & \text{iff}(Prey_v) < f(Prey_u) \end{cases} \end{aligned} \quad (28)$$

where $CP_{i,j}$ represents the carnivorous plant of group i , $Prey_{v,j}$ and $Prey_{u,j}$ respectively represent two randomly selected preys in group i , $growth_{rate}$ is a predefined value, and $rand_{i,j}$ is a random value chosen from the range $[0, 1]$.

The reproduction stage marks the beginning of the exploitation process and is repeated based on the value of n_{CPlant} . In this stage, carnivorous plants use the nutrients obtained to grow and reproduce, thereby ensuring that the optimal individual in the population, that is, the top-ranked individuals, can reproduce. This allows the CPA algorithm to focus on pursuing the optimal solution.

The mathematical expression of this reproduction stage is as follows:

$$\begin{aligned} NewCP_{i,j} &= CP_{1,j} + Reproduction_{rate} \times rand_{i,j} \times mate_{i,j} \\ mate_{i,j} &= \begin{cases} CP_{v,j} - CP_{i,j}, & \text{iff}(CP_i) > f(CP_v) \\ CP_{i,j} - CP_{v,j}, & \text{iff}(CP_i) < f(CP_v) \end{cases}, \quad i \neq v \neq 1 \end{aligned} \quad (29)$$

where $CP_{1,j}$ is the best solution, $CP_{v,j}$ is the randomly selected carnivorous plant, and $Reproduction_rate$ is a predefined value.

The pseudocode of CPA is as follows (Algorithm 1):

Algorithm 1 Carnivorous Plant Algorithm

Input: Objective function $f(x)$, where $x = (x_1, x_2, \dots, x_d)$;
 1: Number of iterations, population size, attraction rate, growth rate, and other parameters.
Output: Best solution found by the algorithm.
 2: Initialize population with n individuals in d dimensions;
 3: Evaluate the fitness of each individual;
 4: Rank individuals according to their fitness values;
 5: Determine the best individual g^* as the first carnivorous plant;
 6: **while** stopping criterion not met **do**
 7: Classify the top n_{CPlant} individuals as carnivorous plants;
 8: Classify the remaining n_{Prey} individuals as prey;
 9: Group the carnivorous plants and prey;
 10: /* Growth of carnivorous plants and prey */
 11: **for** $i = 1$ to n_{CPlant} **do**
 12: **for** Group cycle = 1 to group_iter **do**
 13: **if** attraction_rate > random number **then**
 14: Prey is captured and digested;
 15: Generate a new carnivorous plant using Equation (27);
 16: **else**
 17: Prey escapes;
 18: Generate new prey using Equation (28);
 19: **end if**
 20: **end for**
 21: **end for**
 22: /* Reproduction of the first carnivorous plant */
 23: **for** $i = 1$ to n_{CPlant} **do**
 24: Generate a new carnivorous plant based on the first carnivorous plant using Equation (29);
 25: **end for**
 26: Evaluate the fitness of each new carnivorous plant and prey;
 27: Merge old and newly formed carnivorous plants and prey;
 28: Rank all individuals by fitness;
 29: Determine the current best individual g^* as the first carnivorous plant;
 30: **end while**

4.2. Improved Carnivorous Plant Algorithm

As the number of control points increases, the search efficiency of the Carnivorous Plant Algorithm (CPA) tends to decrease. To address this issue, this paper introduces the following improvements to the CPA: first, a chaotic search strategy is employed to optimize the algorithm's initial solution, enhancing search diversity. Second, a growth stage strategy based on the normal distribution is implemented, allowing the algorithm to dynamically adjust the growth direction during the search process according to the quality of the solution. These enhancements improve both the efficiency and accuracy of the search. The improved algorithm is referred to as the Improved Carnivorous Plant Algorithm (ICPA).

Tent map has become an ideal choice for initializing the population due to its simplicity and efficiency in generating chaotic sequences [19], whose equation is the following:

$$x_{n+1} = \begin{cases} \frac{x_n}{0.7}, & 0 < x_n \leq 0.7 \\ \frac{x_n(1-x_n)}{0.3}, & 0.7 < x_n \leq 1 \end{cases} \quad (30)$$

The specific method of using Tent map to initialize the population is as follows:

$$X_i^j(0) = \begin{cases} x_{min} + Chaos \times (x_{max} - x_{min}), \\ i = 1, 2, \dots, n \\ j = 1, 2, \dots, d \end{cases} \quad (31)$$

where $X_i^j(0)$ represents the initial value of the i -th individual on the j -th decision variable; x_{min} and x_{max} represent the lower and upper bounds of the search space for the independent variables, respectively; M denotes the number of individuals in the population; d is the dimension of the problem considered; and $Chaos$ is the chaos factor generated through the tent map.

In the optimization problem of solving the minimum value, individuals with smaller objective function values are more likely to be close to the optimal solution. To use this characteristic to guide the search direction, we propose a new strategy in which the growth of offspring depends on the normal distribution $N(\mu, \sigma^2)$. Here, μ is the preferred parent, and σ is an adjustable standard deviation that controls the exploration range of new individuals.

According to the 6σ principle of the normal distribution, the interval $(\mu - 3\sigma, \mu + 3\sigma)$ encompasses the majority of possible values. By adjusting σ , we can ensure that the generated offspring are concentrated around μ , thereby enhancing the algorithm's ability to focus on the current best solution. Simultaneously, this approach allows for the exploration of new solutions within a certain range.

To implement this strategy, we replace the previous Equations (27) and (28) with new Equations (32) and (33) during the offspring generation phase:

$$NewCP_{i,j} = N\left(CP_{i,j}, \sqrt{\varepsilon + \left[\frac{CP_{i,j} - Prey_{v,j}}{6} \right]^2} \right) \quad (32)$$

$$NewPrey_{i,j} = \begin{cases} N\left(Prey_{v,j}, \sqrt{\varepsilon + \left[\frac{Prey_{v,j} - Prey_{u,j}}{6} \right]^2} \right) & \text{if } f(Prey_v) > f(Prey_u) \\ N\left(Prey_{u,j}, \sqrt{\varepsilon + \left[\frac{Prey_{u,j} - Prey_{v,j}}{6} \right]^2} \right) & \text{if } f(Prey_v) < f(Prey_u) \end{cases} \quad (33)$$

where ε is a small constant to ensure stability and maintain a certain degree of randomness.

4.3. Smooth and Time-Optimal Trajectory Planning Algorithm

During trajectory optimization, the fitness function evaluates the performance of the solution and guides it to meet objectives and constraints. To handle constraints that may be violated by the solution, we construct a fitness function based on a static penalty approach [20]. Specifically, at each node of the trajectory, we apply functions $P_{constrain}$ to compute various constraints:

$$P_{constrain}(x, x_{min}, x_{max}, C) = C \cdot [\mathbb{I}(x < x_{min}) + \mathbb{I}(x > x_{max})] \quad (34)$$

Here, x represents a certain constrained variable; x_{min} and x_{max} are the minimum and maximum limits of this variable, respectively. C is a fixed positive constant for this type of constraint, and $\mathbb{I}(condition)$ is an indicator function used to check if the condition *condition* is satisfied. $P_{constrain}$ produces a positive value when any constraint violation exists; if there are no violations, it returns 0.

The specific penalty terms constructed from the constraints in the discrete model (17) are as follows:

$$\left\{ \begin{array}{l} P_{\tau}(s_j) = \sum_{i=1}^6 P_{constrain}(\tau_i(s_j, \dot{s}_j, \ddot{s}_j), \tau_{min,i}, \tau_{max,i}, C_{\tau}), \\ P_{\dot{\tau}}(s_j) = \sum_{i=1}^6 P_{constrain}(\dot{\tau}_i(s_j, \dot{s}_j, \ddot{s}_j, \ddot{\tau}_j), \dot{\tau}_{min,i}, \dot{\tau}_{max,i}, C_{\dot{\tau}}), \\ P_{\dot{q}}(s_j) = \sum_{i=1}^6 P_{constrain}(\dot{q}_i(s_j, \dot{s}_j, \ddot{s}_j), \dot{q}_{min,i}, \dot{q}_{max,i}, C_{\dot{q}}), \\ P_{\ddot{q}}(s_j) = \sum_{i=1}^6 P_{constrain}(\ddot{q}_i(s_j, \dot{s}_j, \ddot{s}_j), \ddot{q}_{min,i}, \ddot{q}_{max,i}, C_{\ddot{q}}), \\ P_{\ddot{\ddot{q}}}(s_j) = \sum_{i=1}^6 P_{constrain}(\ddot{\ddot{q}}_i(s_j, \dot{s}_j, \ddot{s}_j), \ddot{\ddot{q}}_{min,i}, \ddot{\ddot{q}}_{max,i}, C_{\ddot{\ddot{q}}}), \\ P_{\dot{s}^*}(s_j) = \sum_{i=1}^6 P_{constrain}(\dot{s}(s_j), 0, \dot{s}_j^*, C_{\dot{s}^*}), \end{array} \right. \quad (35)$$

By summing up the penalty terms for joint constraints at every point along the trajectory, the total penalty value can be determined. Utilizing this value, the fitness function is defined as follows:

$$fitness = J + \sum_{j=1}^n (P_{\tau}(s_j) + P_{\dot{\tau}}(s_j) + P_{\dot{q}}(s_j) + P_{\ddot{q}}(s_j) + P_{\ddot{\ddot{q}}}(s_j) + P_{\dot{s}^*}(s_j)) \quad (36)$$

In summary, the complete algorithm flow for achieving the smooth and time-optimal trajectory planning algorithm (TS-ICPA) of the robot is shown in Figure 6 based on the above discussion.

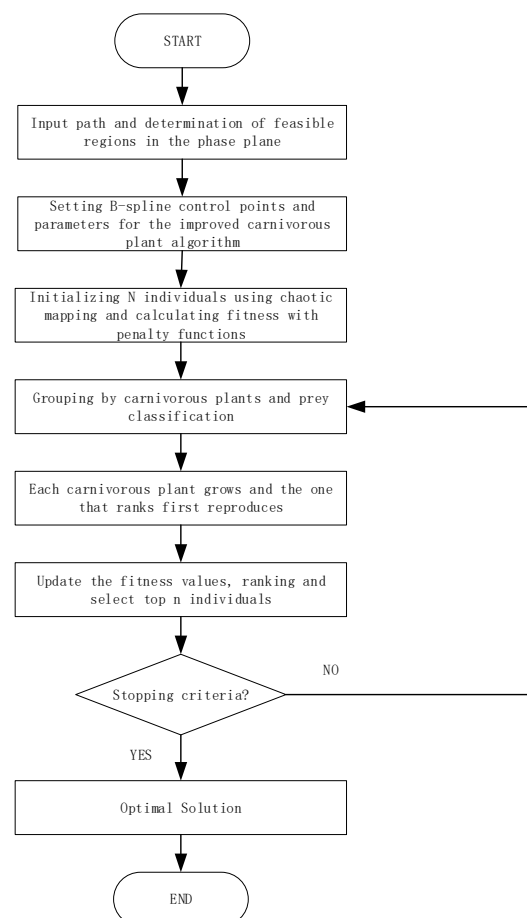


Figure 6. Time-smoothing optimal trajectory algorithm process based on ICPA.

5. Simulation and Result Analysis

5.1. Analysis of Performance Improvement of Carnivorous Plant Algorithm

To assess the optimization performance of the Improved Carnivorous Plant Algorithm (ICPA), this study used six commonly referenced benchmark test functions from the literature [21]. Additionally, we conducted a detailed comparison of the ICPA with several other algorithms, including the original Carnivorous Plant Algorithm (CPA), Genetic Algorithm (GA) [22], Particle Swarm Optimization (PSO) [23], and Artificial Bee Colony (ABC) [24].

This comparison mainly focuses on the performance of each algorithm on the following characteristics: the ability to converge quickly, the effect of identifying a large number of local minima points, the ability to escape from local optimal solutions, and the avoidance of premature convergence.

The detailed information of each test function is listed in Table 1, where F1, F2, and F3 are unimodal benchmark functions, and F4, F5, and F6 are multimodal benchmark functions. For the parameters of the comparison algorithms, we adopted the fine-tuning values recommended in previous work. The parameters of all algorithms are summarized as follows:

- (1) CPA: group iteration = 2; attraction rate = 0.8; growth rate = 2; reproduction rate = 1.8; $n_{CPlant} = 20$; $n_{Prey} = 80$;
- (2) ICAP: group iteration = 2; attraction rate = 0.8; growth rate = 2; reproduction rate = 1.8; $n_{CPlant} = 20$; $n_{Prey} = 80$; Epsilon = 1×10^{-6} ;
- (3) GA: population size = 100; crossover probability, $P_c = 0.8$; mutation probability, $P_m = 0.2$;
- (4) PSO: population size = 100; learning factor, $c = 1.5$; learning factor, $c_2 = 1.5$; inertia weight, $w_{max} = 0.8$; inertia weight, $w_{min} = 0.4$;
- (5) ABC: population size = 100.

Table 1. Set of benchmark functions.

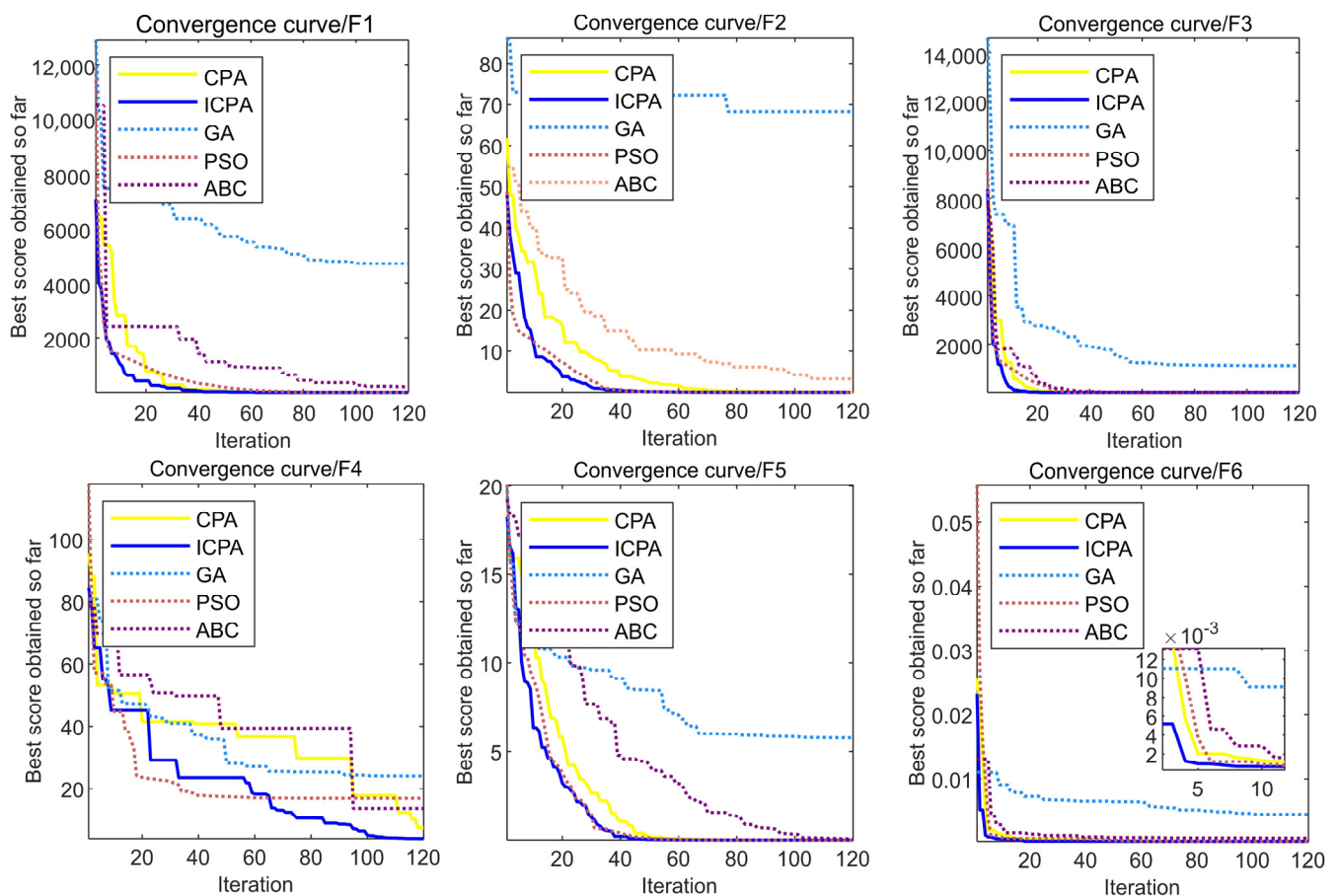
Function	Dim	Range	Optimal
$F_1(x) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2$	10	$[-100, 100]$	0
$F_2(x) = \max_i \{ x_i , 1 \leq i \leq n\}$	10	$[-100, 100]$	0
$F_3(x) = \sum_{i=1}^n ([x_i + 0.5])^2$	10	$[-100, 100]$	0
$F_4(x) = \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i) + 10]$	10	$[-5.12, 5.12]$	0
$F_5(x) = -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right) + 20 + e$	10	$[-32, 32]$	0
$F_6(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2$	4	$[-5, 5]$	0.00030

The statistical significance of the results is ensured and errors are reduced by conducting 30 Monte Carlo runs for each function, with the mean and standard deviation of CPA, ICPA, GA, PSO, and ABC being recorded.

The corresponding test results are presented in Table 2. The optimal iterative convergence curve of each test function is illustrated in Figure 7. It can be observed that ICPA outperforms its comparison algorithm across all test functions.

Table 2. Results of benchmark functions.

Function	Type	CPA	ICPA	GA	PSO	ABC
F1	Mean	1.59×10^0	1.60×10^{-1}	3.85×10^3	4.41×10^{-1}	3.57×10^2
	Std.	9.48×10^{-1}	3.61×10^{-2}	1.98×10^2	6.06×10^{-1}	1.63×10^2
F2	Mean	3.46×10^{-2}	1.77×10^{-3}	2.45×10^1	3.04×10^{-2}	4.54×10^0
	Std.	1.98×10^{-3}	5.34×10^{-4}	2.14×10^0	1.65×10^{-2}	7.61×10^{-1}
F3	Mean	1.38×10^{-5}	2.29×10^{-8}	1.19×10^2	5.39×10^{-6}	3.38×10^{-2}
	Std.	3.67×10^{-6}	5.44×10^{-9}	4.15×10^1	7.30×10^{-6}	5.31×10^{-4}
F4	Mean	2.24×10^1	4.64×10^0	2.94×10^1	1.53×10^1	3.22×10^1
	Std.	7.55×10^0	6.74×10^{-1}	9.43×10^0	9.04×10^0	5.10×10^0
F5	Mean	1.11×10^{-3}	3.98×10^{-5}	9.30×10^0	8.23×10^{-1}	1.61×10^{-1}
	Std.	3.41×10^{-5}	1.28×10^{-5}	5.50×10^0	1.16×10^0	1.67×10^{-2}
F6	Mean	3.09×10^{-4}	3.07×10^{-4}	6.65×10^{-3}	3.08×10^{-4}	7.31×10^{-4}
	Std.	1.51×10^{-6}	8.06×10^{-8}	5.54×10^{-3}	9.59×10^{-16}	5.27×10^{-6}

**Figure 7.** Atlas of optimal iteration convergence curves for benchmark functions.

5.2. Simulation Experiments and Analysis

A trajectory optimization simulation experiment was conducted on the UR5 collaborative robot along a predetermined path. Figure 8 shows the UR5 collaborative robot used in the simulation experiment. The DH parameters are used to calculate the kinematics and dynamics of the UR5 robot, with its coordinate system and associated parameters illustrated in Figure 9 and Table 3.

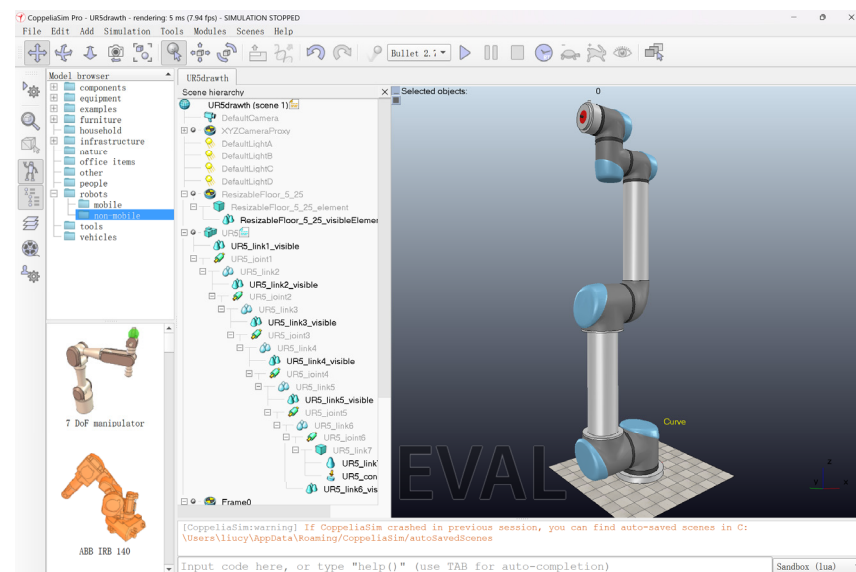


Figure 8. Graphical result of UR5 modelled in Coppeliasim.

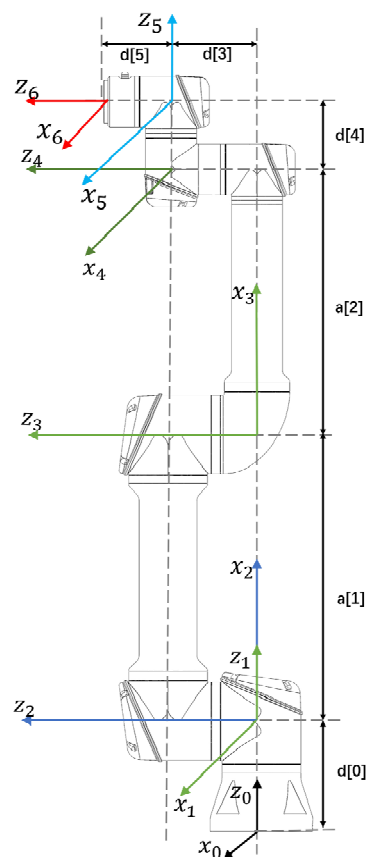


Figure 9. Schematic and frames assignment of UR5.

Table 3. The DH parameters used to calculate the UR5 kinematics and dynamics.

Kinematics	alpha [rad]	a [m]	d [m]	theta [rad]
Joint 1	$\pi/2$	0	0.089159	0
Joint 2	0	−0.425	0	$\pi/2$
Joint 3	0	−0.39225	0	0
Joint 4	$\pi/2$	0	0.10915	$-\pi/2$
Joint 5	$-\pi/2$	0	0.09465	0
Joint 6	0	0	0.0823	0

In the experiments of this paper, Matlab and CoppeliaSim (formerly V-REP) were mainly used as simulation platforms. The control code was written in C++. The experiments were run in the ROS environment, and the hardware platform was a computer equipped with an Intel(R) Xeon(R) i5-7500 3.4 GHz processor and 16 GB memory. With this hardware and software configuration, the experiment was able to achieve efficient simulation and data analysis of robot motion control, verifying the effectiveness and practical applicability of the algorithm.

Table 4 presents the required configurations and motion limits for each robot joint, including constraints on velocity, acceleration, and other key parameters. These limits define the safe operating range for each joint, ensuring that the robot's motion stays within feasible boundaries. Based on the physical and mechanical specifications of the robot's joints, these constraints are crucial for maintaining operational stability and ensuring smooth performance.

Table 4. UR5 experimental boundary conditions.

Joint <i>i</i>	1	2	3	4	5	6
Velocity (deg/s)	160	160	160	160	160	160
Acceleration (deg/s ²)	573	573	573	1146	1146	1146
Jerk(deg/s ³)	10,000	10,000	10,000	10,000	10,000	10,000
Torque (Nm)	100	100	100	150	150	150
Torque rate(Nm/s)	500	500	500	500	500	500

The generation process for the test path is as follows: firstly, five nodes are randomly selected within the range of $[-80^\circ, 80]^\circ$, and from these nodes, 200 equidistant nodes are generated. Then, cubic spline interpolation is applied to these equidistant nodes to form the joint space path $q(s)$.

The positions of the five initial nodes used in the experiment are shown in Table 5. The s path is defined by $ds = \|dq\|$. To ensure high accuracy of the trajectory solution, the test path was resampled before the experiment with an s resolution of 0.293, thereby transforming it into a dense s path containing 2400 nodes.

Table 5. Five initial node positions of the test path.

Position No.	Joint No. (°)					
	1	2	3	4	5	6
1	−27.16	−1.13	−66.75	6.31	17.22	−32.13
2	33.52	−11.10	−70.92	−44.91	−52.19	−13.37
3	57.52	32.53	−34.89	−75.99	55.31	−28.93
4	−6.75	−54.11	−10.69	−57.08	69.62	58.80
5	−23.62	68.46	−50.35	65.07	−78.60	56.47

In this paper, we explore an innovative trajectory planning method based on phase plane theory. This method adopts the comprehensive optimization of total jerk and time as the core goal while considering complex kinodynamic constraints up to the third order.

Currently, there are no similar studies that consider these factors comprehensively, which makes comparative analysis of our approach with existing studies challenging. To demonstrate the effectiveness and advantages of our proposed method, this study compares two approaches through time-optimal simulation experiments. The first is our newly developed method, TS-ICPA (where the time-jerk optimization weight λ is set to 0), and the second is the Advanced TOPP-BA, which employs second-order kinodynamic constraints.

The number of B-spline control points of our trajectory optimization algorithm is 120, the upper limit of ICPA iterations is 200, and the penalty coefficient is set to 50. Apart from this, other experimental parameters remain consistent with those in the performance test.

Figure 10 shows the time parameterization results generated by the TOPP-BA using second-order kinodynamic constraints and our method ($\lambda = 0$) on the test path. Preliminary observations show that the two curves behave fairly closely in most cases, and the relative errors with each other remain within a small range. The results show a high correlation of 0.986. This numerical analysis, along with our observations, suggests that while there are some minor differences between the two algorithms, they generally produce very similar outcomes overall.

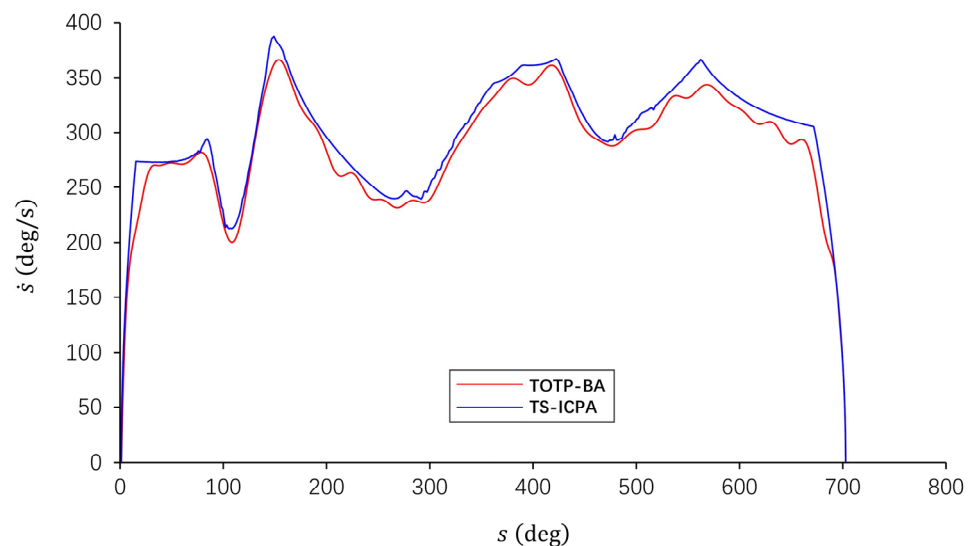


Figure 10. TOPP—BA and TS—ICPA ($\lambda = 0$) time parameterization results compared.

For further analysis, Figures 10 and 11 present the normalized trajectory results for each joint. The constant-step fifth-order Dormand–Prince method, an explicit Runge–Kutta ODE solver, is applied to process the time-parameterized curve. To ensure accuracy, an integration step size of 0.008 s was used during the processing.

As shown in Figure 11, the joint velocity, acceleration, and torque trajectories of the two methods are highly similar in terms of overall shape, peak positions, and proximity to the set boundaries, which further supports the validity of our approach. In theory, both methods should result in the same motion time; however, our method exhibited a slight increase of 0.31 s. Despite this, the trajectories generated by our algorithm demonstrate greater smoothness and consistency compared to the reference TOPP-RA.

Two key factors contribute to this outcome. First, our method is heuristic, meaning it asymptotically approaches the optimal solution rather than directly finding the global optimum. Second, unlike TOPP-RA, our method incorporates additional constraints, specifically on jerk and torque rate, which enhances the smoothness and control of the generated trajectories.

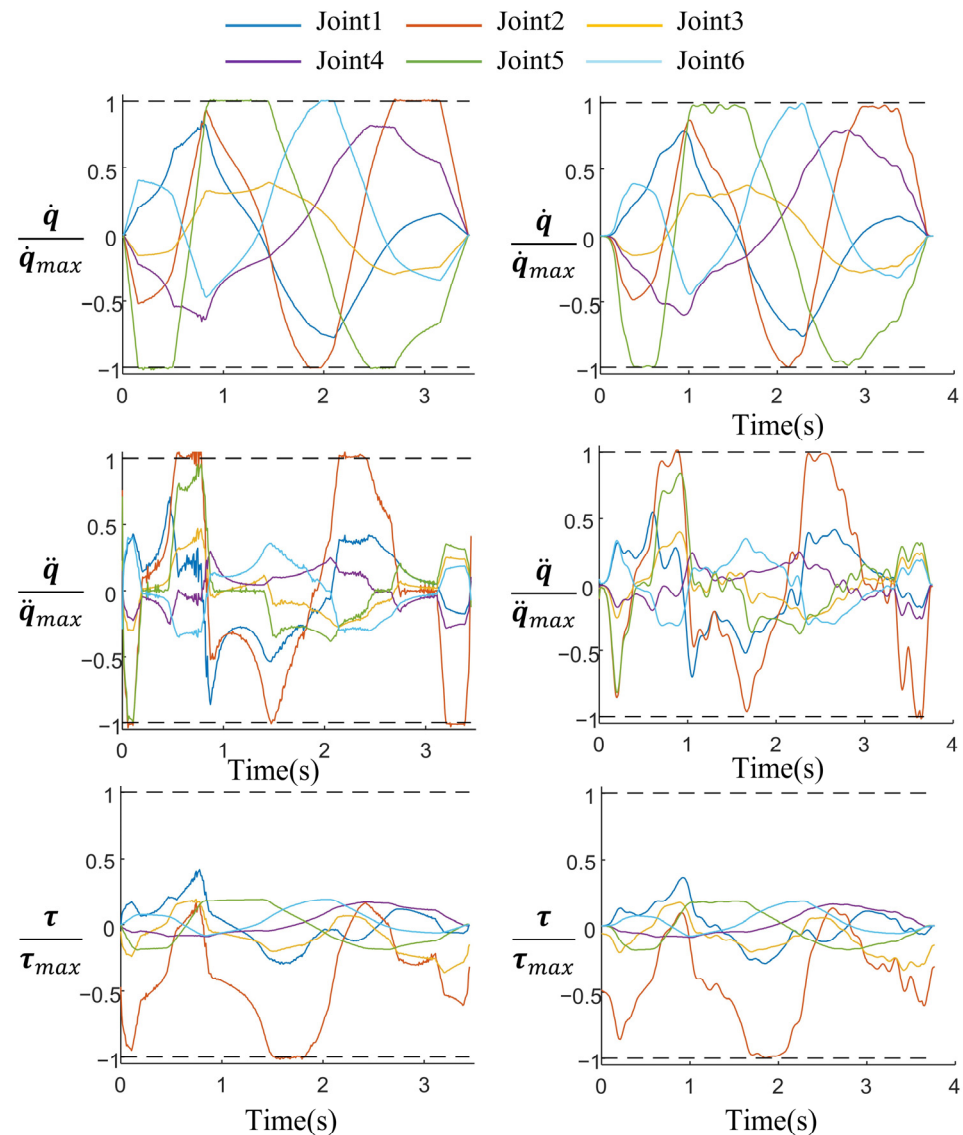


Figure 11. (a) Normalized results of TOTP-BA; (b) normalized results of TS-ICPA.

The application of the TS-ICPA method results in improved joint motion stability, as illustrated in Figure 12. A comparative analysis of jerk and torque rate variations reveals that with the TOPP-BA approach (Figure 12a,c), the robot's normalized joint jerk falls within the range of $(-14.8, 6.0)$, while the normalized torque rate ranges from $(-6.1, 17.4)$. In contrast, when the TS-ICPA method is applied (Figure 12b,d), both the normalized squared jerk and torque rate are significantly reduced and confined within the narrower range of $(-1, 1)$, demonstrating the method's effectiveness in enhancing motion smoothness and stability.

In the absence of additional constraints, the TOPP-BA method results in significant variations in jerk and torque rates along certain path segments. For example, Figure 12a demonstrates substantial fluctuations in the jerk of joint 1 between $t = 0.59$ s and $t = 0.88$ s, while Figure 12c shows marked variations in the torque rate of joint 2 between $t = 0.68$ s and $t = 0.78$ s. In contrast, our method introduces third-order motion dynamics constraints and employs cubic spline interpolation to smooth these abrupt changes. This approach effectively reduces the peak values of the parameter trajectories and enhances the smoothness of motion, as evidenced in Figure 12b,d. Although this method results in a slight increase in motion time, it significantly improves stability.

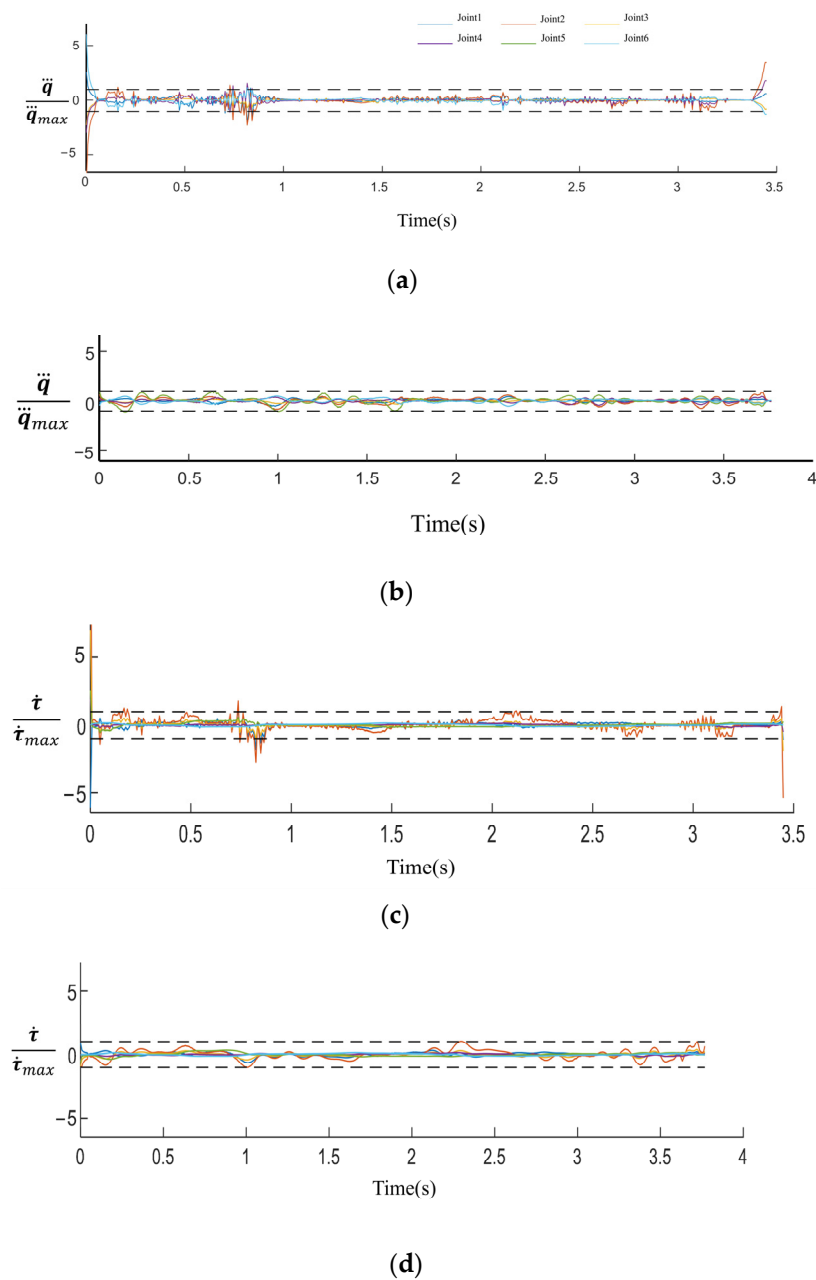


Figure 12. (a) The normalized jerk results for TOTP-BA. (b) The normalized jerk results for TS-ICPA. (c) The normalized torque rate results for TOTP-BA. (d) The normalized torque rate results for TS-ICPA.

Figure 13 illustrates the relationship between the trajectory planning objectives—time efficiency and motion smoothness—as the trade-off coefficient λ takes on different values. This demonstrates how varying λ impacts the balance between minimizing the trajectory duration and enhancing the smoothness of the motion.

For an intuitive comparison with the case where $\lambda = 0$, both the trajectory duration and the total squared jerk have been normalized relative to their values at $\lambda = 0$. When the trade-off coefficient λ is set to 0.5, 1, and 1.5, the corresponding normalized jerk results for the trajectories generated by the proposed method are presented in Figure 14. This comparison highlights how adjusting λ influences the balance between trajectory duration and smoothness.

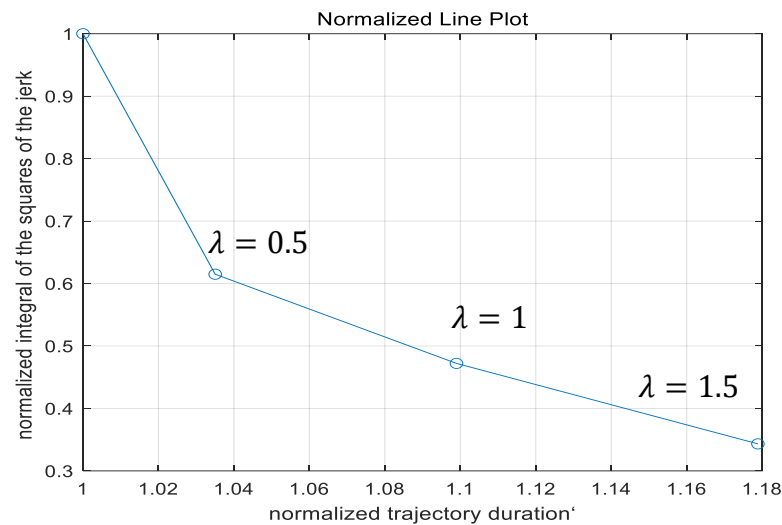


Figure 13. Relationship between normalized trajectory duration and normalized total squared jerk.

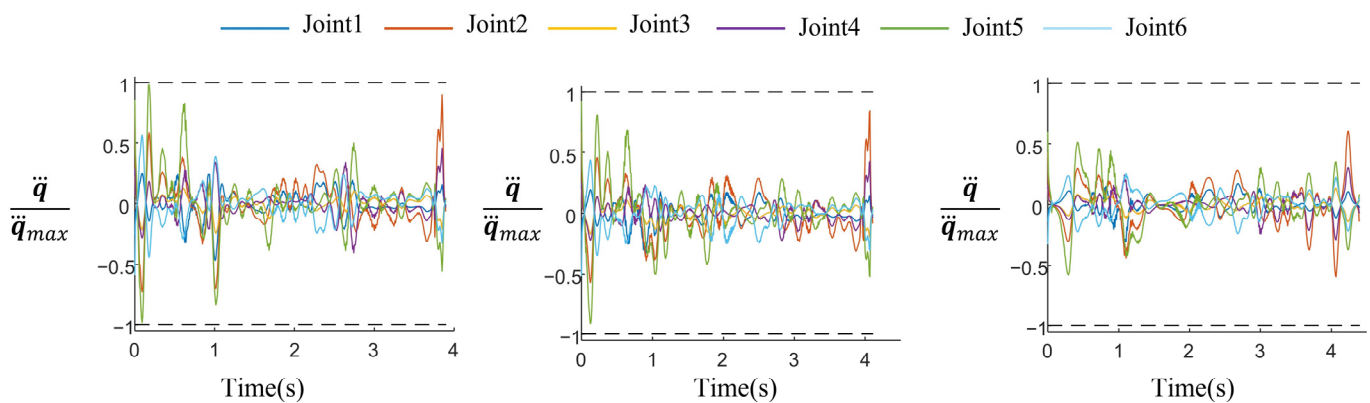


Figure 14. (a) The results when λ equals 0. (b) The results when λ equals 0.5. (c) The results when λ equals 1.5.

The results in Figure 13 indicate that as the trade-off coefficient λ increases from 0 to 0.5, the trajectory duration rises by 3.5% (reaching approximately 1.035), while the normalized total squared jerk decreases by about 38% (to approximately 0.62). As λ increases further to 1, the trajectory duration increases by an additional 9.8% compared to the baseline, with the normalized total squared jerk decreasing by around 54%. When λ reaches 1.5, the trajectory duration increases by 17.9%, accompanied by a significant reduction in normalized total squared jerk by about 65%.

From Figure 14, it can be inferred that as λ increases, the acceleration range for each joint contracts. For example, joint 1's minimum acceleration decreases from -0.465 at $\lambda = 0.5$ to -0.351 at $\lambda = 1.5$, while its maximum acceleration reduces from 0.247 to 0.171 . This suggests that increasing λ leads to smoother motion at the cost of longer trajectory duration, as reflected by the reduced acceleration extremes.

The numerical results align with our engineering intuition: achieving time efficiency often requires making compromises in motion smoothness. These findings highlight the adaptability and flexibility of our proposed trajectory optimization model, which can dynamically adjust the trade-off coefficient to meet specific requirements. This allows the model to strike an optimal balance between high-speed motion and smoothness. The demonstrated optimization strategy is highly valuable in scenarios where both motion efficiency and stability are critical considerations.

6. Conclusions

This paper introduces a novel approach for smooth and time-optimal trajectory planning of robots. The method centers on constructing an integrated objective function in the phase plane, which accounts for both the total execution time and the squared integral of acceleration. By optimizing these factors, the approach ensures that the robot's motion is both efficient and smooth throughout the trajectory.

After formulating the objective function, this paper performs a comprehensive analysis of the kinematic and dynamic constraints up to the third order in the phase plane to define the feasible trajectory region. To further enhance trajectory smoothness and improve computational efficiency, a third-order B-spline interpolation method was utilized for trajectory generation.

Additionally, an enhanced Carnivorous Plant Algorithm was employed to optimize the trajectories, demonstrating notable efficiency in the optimization process. The conducted simulation experiments validate the effectiveness and superiority of the proposed method, showing its ability to generate smooth, time-optimal trajectories under constrained conditions.

Author Contributions: Conceptualization, X.Z.; formal analysis, B.W.; funding acquisition, B.W.; investigation, K.Z. and Z.C. (Zhengfeng Cao); methodology, C.L. and X.Z.; project administration, B.W.; resources, B.W. and Z.C. (Zhengfeng Cao); software, C.L. and X.Z.; supervision, B.W., K.Z., Z.C. (Zhengfeng Cao) and Z.C. (Zexin Chen); validation, C.L., X.Z. and Z.C. (Zexin Chen); visualization, C.L. and X.Z.; writing—original draft, C.L. and X.Z.; writing—review and editing, C.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by scientific and technological research program of chongqing municipal education commission: KJZD-K202400607 and special projects for technological innovation and application development in chongqing municipality: CSTC2021JSCX-GKSBX0055.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lin, C.; Chang, P.; Luh, J. Formulation and optimization of cubic polynomial joint trajectories for industrial robots. *IEEE Trans. Autom. Control.* **1983**, *28*, 1066–1074. [\[CrossRef\]](#)
2. Gasparetto, A.; Zanotto, V. A new method for smooth trajectory planning of robot manipulators. *Mech. Mach. Theory* **2007**, *42*, 455–471. [\[CrossRef\]](#)
3. Gasparetto, A.; Zanotto, V. A technique for time-jerk optimal planning of robot trajectories. *Robot. Comput.-Integr. Manuf.* **2008**, *24*, 415–426. [\[CrossRef\]](#)
4. Wang, F.; Wu, Z.; Bao, T. Time-jerk optimal trajectory planning of industrial robots based on a hybrid woa-ga algorithm. *Processes* **2022**, *10*, 1014. [\[CrossRef\]](#)
5. Machmudah, A.; Parman, S.; Zainuddin, A.; Chacko, S. Polynomial joint angle arm robot motion planning in complex geometrical obstacles. *Appl. Soft Comput.* **2013**, *13*, 1099–1109. [\[CrossRef\]](#)
6. Hou, R.; Niu, J.; Guo, Y.; Ren, T.; Yu, X.; Han, B.; Ma, Q. A novel resolution scheme of time-energy optimal trajectory for precise acceleration controlled industrial robot using neural networks. *Actuators* **2022**, *11*, 130. [\[CrossRef\]](#)
7. Bobrow, J.E.; Dubowsky, S.; Gibson, J.S. Time-optimal control of robotic manipulators along specified paths. *Int. J. Robot. Res.* **1985**, *4*, 3–17. [\[CrossRef\]](#)
8. Shin, K.; McKay, N. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Trans. Autom. Control.* **1985**, *30*, 531–541. [\[CrossRef\]](#)
9. Shiller, Z.; Lu, H.H. Computation of path constrained time optimal motions with dynamic singularities. *J. Dyn. Sys., Meas., Control* **1992**, *114*, 34–40. [\[CrossRef\]](#)
10. Verschuere, D.; Demeulenaere, B.; Swevers, J.; De Schutter, J.; Diehl, M. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Trans. Autom. Control.* **2009**, *54*, 2318–2327. [\[CrossRef\]](#)
11. Oberherber, M.; Gattringer, H.; Müller, A. Successive dynamic programming and subsequent spline optimization for smooth time optimal robot path tracking. *Mech. Sci.* **2015**, *6*, 245–254. [\[CrossRef\]](#)
12. Ham, H.; Pham, Q.C. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Trans. Robot.* **2018**, *34*, 645–659.
13. Barnett, E.; Gosselin, C. A bisection algorithm for time-optimal trajectory planning along fully specified paths. *IEEE Trans. Robot.* **2020**, *37*, 131–145. [\[CrossRef\]](#)

14. Lin, H.I. A fast and unified method to find a minimum-jerk robot joint trajectory using particle swarm optimization. *J. Intell. Robot. Syst.* **2014**, *75*, 379–392. [[CrossRef](#)]
15. Gao, X.; Mu, Y.; Gao, Y. Optimal trajectory planning for robotic manipulators using improved teaching-learning-based optimization algorithm. *Ind. Robot.* **2016**, *43*, 308–316. [[CrossRef](#)]
16. Huang, J.; Hu, P.; Wu, K.; Zeng, M. Optimal time-jerk trajectory planning for industrial robots. *Mech. Mach. Theory* **2018**, *121*, 530–544. [[CrossRef](#)]
17. Sencer, B.; Altintas, Y.; Croft, E. Feed optimization for five-axis cnc machine tools with drive constraints. *Int. J. Mach. Tools Manuf.* **2008**, *48*, 733–745. [[CrossRef](#)]
18. Ong, K.M.; Ong, P.; Sia, C.K. A carnivorous plant algorithm for solving global optimization problems. *Appl. Soft Comput.* **2021**, *98*, 106833. [[CrossRef](#)]
19. Javidi, M.; Hosseinpourfard, R. Chaos Genetic Algorithm Instead Genetic Algorithm. *Int. Arab. J. Inf. Technol.* **2015**, *12*, 163–168.
20. Mezura-Montes, E.; Coello, C.A.C. Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm Evol. Comput.* **2011**, *1*, 173–194. [[CrossRef](#)]
21. Jamil, M.; Yang, X.S. A literature survey of benchmark functions for global optimisation problems. *Int. J. Math. Model. Numer. Optim.* **2013**, *4*, 150–194. [[CrossRef](#)]
22. Deng, W.; Zhang, X.; Zhou, Y.; Liu, Y.; Zhou, X.; Chen, H.; Zhao, H. An enhanced fast non-dominated solution sorting genetic algorithm for multi-objective problems. *Inf. Sci.* **2022**, *585*, 441–453. [[CrossRef](#)]
23. Li, G.; Chou, W. Path planning for mobile robot using self-adaptive learning particle swarm optimization. *Sci. China Inf. Sci.* **2018**, *61*, 1–18. [[CrossRef](#)] [[PubMed](#)]
24. Bhattacharjee, P.; Rakshit, P.; Goswami, I.; Konar, A.; Nagar, A.K. Multi-robot path-planning using artificial bee colony optimization algorithm. In Proceedings of the Third World Congress on Nature and Biologically Inspired Computing, Salamanca, Spain, 19–21 October 2011; pp. 219–224.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.