Mathematical Biosciences and Engineering

MBE, 21(2): 3364–3390.
DOI: 10.3934/mbe.2024149
Received: 16 November 2023
Revised: 16 January 2024
Accepted: 19 January 2024
Published: 04 February 2024

http://www.aimspress.com/journal/MBE

Research article

A trajectory planning method for a casting sorting robotic arm based on a nature-inspired Genghis Khan shark optimized algorithm

Chengjun Wang[1,2], Xingyu Yao[2,*], Fan Ding[2], and Zhipeng Yu[1]

[1] School of Artificial Intelligence, Anhui University of Science and Technology, Huainan 232001, China
[2] School of Mechanical Engineering, Anhui University of Science and Technology, Huainan 232001, China

* **Correspondence:** Email: 2021200691@aust.edu.cn.

Abstract: In order to meet the efficiency and smooth trajectory requirements of the casting sorting robotic arm, we propose a time-optimal trajectory planning method that combines a heuristic algorithm inspired by the behavior of the Genghis Khan shark (GKS) and segmented interpolation polynomials. First, the basic model of the robotic arm was constructed based on the arm parameters, and the workspace is analyzed. A matrix was formed by combining cubic and quintic polynomials using a segmented approach to solve for 14 unknown parameters and plan the trajectory. To enhance the smoothness and efficiency of the trajectory in the joint space, a dynamic nonlinear learning factor was introduced based on the traditional Particle Swarm Optimization (PSO) algorithm. Four different biological behaviors, inspired by GKS, were simulated. Within the premise of time optimality, a target function was set to effectively optimize within the feasible space. Simulation and verification were performed after determining the working tasks of the casting sorting robotic arm. The results demonstrated that the optimized robotic arm achieved a smooth and continuous trajectory velocity, while also optimizing the overall runtime within the given constraints. A comparison was made between the traditional PSO algorithm and an improved PSO algorithm, revealing that the improved algorithm exhibited better convergence. Moreover, the planning approach based on GKS behavior showed a decreased likelihood of getting trapped in local optima, thereby confirming the effectiveness of the proposed algorithm.

Keywords: casting sorting manipulator; Genghis Khan shark optimizer; trajectory planning; polynomial interpolation; PSO algorithm

1. Introduction

With the continuous development of modern industry and the improvement of automation level, more and more industrial robots are applied in modern manufacturing. In order to pursue a more efficient and stable robot arm, trajectory planning is one of the important parameters to evaluate the motion of the robot arm. At present, a lot of research has been carried out for the smooth trajectory, optimal time, shortest path and lowest energy consumption of robotic arms. There are different constraints for different task requirements. In the casting sorting industry, for the handling and sorting of small castings with a large number of sand falls, there are certain requirements for the handling speed and smoothness of the mechanical arm, and time is an important factor to be considered to ensure that the execution efficiency of the casting sorting robot is maximized within the allowed range.

Generally, the motion path points used for trajectory planning are given in Cartesian space or joint space, thus ensuring that all target positions are reachable during the robotic arm is functioning. Under the constraint condition, it is the main goal to find a trajectory line with the shortest time between multi-segment waypoints. In recent years, a plethora of methods have been studied to optimize trajectories. Smooth trajectories can be constructed using interpolation functions, such as spline functions and polynomial functions [1–6]. In existing studies, it has been proposed to use 3rd-degree polynomial and Bessel curve to compound smooth trajectories [7], and effectively reduced the speed and acceleration. However, compared with B-spline curves, Bessel curve lacks local modification, and the trajectory optimized by non-uniform B-spline curves can adjust the curvature of local curves [8]. To ensure acceleration continuity, a method of generating trajectories involves segmenting and discretizing the joint angle sequence using fifth-order polynomial interpolation [9]. Zhao et al. conducted kinematic analysis and simulation comparisons of 5th order and 3rd order polynomials to validate the planning effectiveness of the 5th order polynomial through five iterations of verification. This approach was then applied to six types of industrial robotic arms [10]. Smooth trajectories for the robotic arm can also be achieved through the combination of traditional interpolation functions. By utilizing segment-wise cubic interpolation functions, such as Hermite-type functions, that also can be generated smooth trajectories [11,12]. Alternatively, a novel interpolation planning technique called "5-4-5" or "5-4-5-5" can be employed. By arranging different segments of polynomial interpolation function sequences, the smoothness and stability of the trajectories can be significantly improved [13].

These methods improve the motion performance of the robot through interpolation or curvature calculations. However, to ensure both smooth trajectories and minimal execution time, adjustments need to be made to the motion parameters and the methods employed for generating smooth trajectories. Generally, the trajectory planning problem is transformed into nonlinear function optimization problem by intelligent optimization algorithm. In the realm of optimization, there are two major directions: Deterministic and non-deterministic. Deterministic algorithms, often gradient-based, have some optimization effectiveness for linear or nonlinear function problems. Traditional heuristic algorithms also fall into this category. However, due to their fixed inputs and outputs, as well as repetitive computations, they are prone to becoming trapped in local optima [14]. On the other hand,

non-deterministic methods rely on a combination of randomization and heuristic solutions, such as genetic algorithms and particle swarm optimization (PSO). These methods leverage their inherent randomness, diversity, and independence to break free from the vicious cycle of local optima and effectively solve and optimize problems. In recent years, many scholars have adopted these methods to optimize trajectory planning problems. Using a genetic algorithm, Zhang et al. conducted a global optimization of fifth-order interpolation joint nodes and computed the optimal time intervals between the interpolation points of each joint [15]. Shi et al. proposed an improved simulated annealing algorithm to overcome challenges associated with high computational complexity and local minima. This algorithm was designed for dynamic path planning, aiming to provide time-optimal solutions for the robotic arm in both static and dynamic environments [16]. Zhang et al. adopted Sparrow Search Algorithm (SSA) based on Tent Chaos mapping to achieve real-time optimization of robot arm trajectory [17]. Wang et al. introduced an enhanced Whale Optimization Algorithm to optimize the objective function defined by time and pulses. This approach ensures the smooth continuity and operational efficiency of the robot [18]. Ivan Carvajal et al. optimized the path length of a 6-axis robot based on nonlinear regression of cubic polynomial combinations to represent trajectories. They validated the feasibility of this approach through simulation [19]. Özge and Bekir utilized the Particle Swarm Optimization (PSO) algorithm to optimize fifth-order polynomial interpolation functions. Their approach specifically aimed to determine the shortest path for point-to-point tasks and achieve time optimization [20]. There are many other optimization algorithms and joint trajectory planning methods for different types of robots, not limited to a single algorithm or approach. For different robots and different optimization goals, there has been considerable relevant research [21–23]. In terms of bio-inspired optimization algorithms, with the continuous emergence of non-deterministic intelligent optimization algorithms, such as genetic algorithms, particle swarm algorithms, or fish swarm algorithms, the field has embarked on analyzing the behaviors of various organisms and deriving relevant mathematical models. This has resulted in a diverse range of bio-inspired optimization algorithms, including the whale algorithm, zebra algorithm, and social spider algorithm. These algorithms have shown promise in addressing conventional engineering application problems, but they also come with their own strengths and weaknesses. Some intelligent optimization algorithms may suffer from premature convergence, lack of robust global search capabilities, or a tendency to become trapped in local optima. However, a novel nature-inspired heuristic algorithm proposed by Gang Hu and colleagues, based on the behavior of Genghis Khan sharks, offers a unique approach [24]. By emulating the hunting, movement, foraging, and self-protection behaviors of GKS, this algorithm effectively executes optimization tasks for intelligent agents and accomplishes desired optimization objectives. While there are numerous case studies analyzing specific mathematical problems related to this method, there is limited research on its application in practical robotic arm trajectory planning. Hence, this article holds significant research value in exploring the potential of this approach in such real-world applications.

We present a method for optimizing the trajectory of a casting sorting robotic arm using a nature-inspired heuristic algorithm based on the behavior of Genghis Khan sharks. Calculating the basic parameters of the robotic arm for casting sorting through robot kinematics. With the constraints of task path angles, velocity, and acceleration in mind, a segmented polynomial interpolation method called "3-5-3" is utilized for joint trajectory planning. Objective functions are then constructed, and the optimization problem is solved using an equivalent mathematical model that mimics the four behaviors of GKS. The algorithm seeks to find the optimal time solution for the trajectory problem. The obtained

results are compared with those from particle swarm optimization and improved particle swarm optimization algorithms. To validate the effectiveness of the algorithm, experimental simulations are conducted.

2. Robot kinematics

2.1. Kinematic calculations

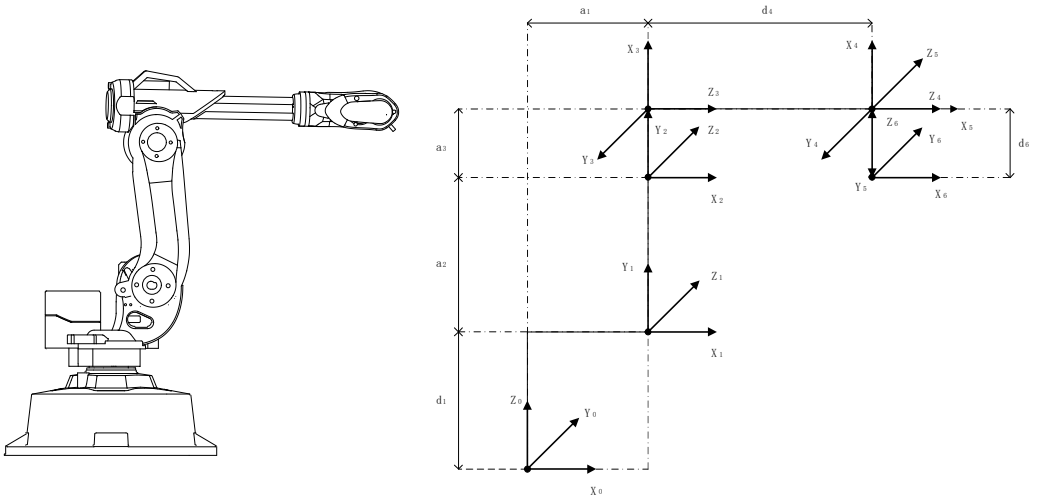


Figure 1. Six-axis robotic arm coordinate system.

The robotic arm consists of a series of rotating or translating joints combined with multiple linkages. By manipulating the angles between the joints, the desired pose of the end effector can be determined. Kinematic calculations involve transforming between joint angle space and Cartesian space to establish the relationship between angles and positions. The coordinate system of the mechanical arm is established with the rotating joint of the mechanical arm as the *Z* axis. Figure 1 shows the relationship between the coordinate systems on the six-axis mechanical arm. Through the Angle and displacement parameters of each connecting rod, the position and attitude of the end can be described by the transformation matrix.

Table 1. Denavit-Hartenberg table of 6-DOF robotic arm.

Link number	Link twist angle α_{i-1} (rad)	Link length a_{i-1} (mm)	Joint distance d_i (mm)	Joint angle θ_{i-1} (rad)
1	-pi/2	32	80	0
2	0	108	0	- pi/2
3	-pi/2	20	0	0
4	pi/2	0	175	0
5	-pi/2	0	0	0
6	0	0	20	0

Through forward kinematics, joint angles and distances can be converted into an end effector pose

transformation matrix. Inverse kinematics calculations can transfer the joint angles, based on this matrix. By utilizing the provided parameters, the kinematic equations for the robot can be established

1) Inter-bar transformation matrix

$$ {}^{i-1}_{i}T = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

2) The kinematic equation is established by multiplying multiple transformation matrices to obtain the transformation matrix of the angle and displacement of the end effector.

$$ {}^{0}_{6}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T\,{}^{3}_{4}T\,{}^{4}_{5}T\,{}^{5}_{6}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

where $r_{i,j}$ $(i = 1, 2, 3, j = 1, 2, 3)$ is the rotation Angle transformation matrix and P_x, P_y, P_z is the displacement transformation matrix. When the end attitude is given, the joint Angle is inversely solved using the inverse transformation of the connecting rod matrix. The inverse solution obtained is typically not unique. By comparing the motion parameters of each joint and sequentially eliminating possibilities, the correct angles can be determined. Axes 1, 2, and 3 are responsible for determining the precise positioning of the robot hand within the operational area, while axes 4, 5, and 6 play a crucial role in establishing the desired orientation of the end effector. Algebraic methods are employed to solve for the first three axes. By left-multiplying the equation ${}^{i}_{i-1}T$ with the inverse transformation matrix, the joint variables are separated, enabling their determination. By gradually solving for θ_1 to θ_2 using element-wise comparisons, there can be up to 8 possible solutions. However, due to structural limitations and constraints on joint angles, some solutions can be ruled out. In situations with multiple solutions, a suitable set of solutions can be selected based on specific criteria, such as minimizing joint motion or ensuring motion accuracy [25–28]. This selection process aims to fulfill the operational requirements of the robotic arm.

2.2. Robotic workspace

The dimensions of a small-scale casting sorting robotic arm are determined based on the desired working space required for factory tasks. This workspace includes the region where the robotic arm can effectively grip and place small castings. The working space of the robotic arm is defined by considering the limitations on joint angles, as well as the constraints on velocity and acceleration. These limitations and constraints are outlined in Table 2.

Table 2. Robotic arm parameter limits.

Joint	1	2	3	4	5	6
Working range (rad)	-1.92~2.79	-1.75~1.75	-2.09~1.05	-3.14~3.14	-3.49~2.09	-6.28~6.28
Maximum speed (rad/s)	1.48	1.05	1.13	3.49	3.49	7.85
Maximum Acceleration (rad^2/s)	0.87	0.78	0.78	1.31	1.31	1.40

Traditional graphical methods and numerical methods have limitations in accurately describing the working space of a mechanical system. Numerical methods often involve excessive computational complexity, and their results may not be sufficiently reliable for regions near the boundaries. Therefore, a statistical simulation method called the Monte Carlo method is utilized to construct the comprehensive working space of the robotic arm as accurately as possible [29]. This approach relies on the joint angle limitations for solving the problem.

The fundamental principle of this solution is to randomly sample angles within the range of joint angles for the robotic arm and use the accumulation of these random points to construct the overall working space. This method is applicable to the calculation of the working space for robotic arms with various joint configurations. In this study, a total of 30,000 random points is generated. Figure 2 depicts the working space of the small-scale casting sorting robotic arm.

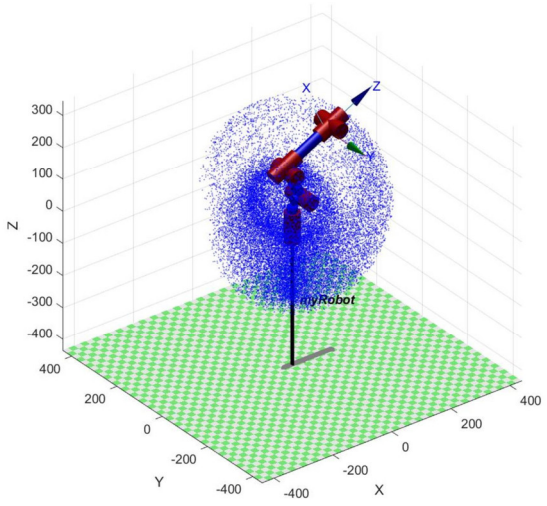


Figure 2. Casting sorting robotic arm workspace.

3. Trajectory planning

3.1. Cubic polynomial interpolation function

The point-to-point motion mode (PTP) of a small casting sorting robot can utilize various common polynomial interpolation functions, such as cubic polynomials, fifth-degree polynomials, seventh-degree polynomials, etc. In situations where the initial and end points have zero velocity, the third-order polynomial enables interpolation between two points to create intermediate points while also defining the joint angle and time. Therefore, we adopt third-order polynomials. By sequentially

processing these intermediate points, the following expression is obtained:

$$q(t) = a_0 + a_1(t-t_0) + a_2(t-t_0)^2 + a_3(t-t_0)^3 \tag{3}$$

In this equation, a_i (where $i = 0, 1, 2, 3$) represents the coefficients of the polynomial. Taking the derivative of the polynomial allows us to derive the expressions for the joint velocity and angular acceleration of the robotic arm:

$$\begin{cases} \dot{q}(t) = a_1 + 2a_2(t-t_0) + 3a_3(t-t_0)^2 \\ \ddot{q}(t) = 2a_2 + 6a_3(t-t_0) \end{cases} \tag{4}$$

To meet the constraints of joint angle limitations and angular velocity restrictions at the initial and final time instants, it is possible to set constraints and solve for the four unknown coefficients in the cubic polynomial.

$$\begin{cases} q(t_0) = q_0 \\ q(t_f) = q_f \\ \dot{q}(t_0) = \dot{q}_0 \\ \dot{q}(t_f) = \dot{q}_f \end{cases} \tag{5}$$

where t_0 and t_f represent the initial and final times for each segment, q_0 and q_f represent the initial and final angles, $\dot{q}_0$ and $\dot{q}_f$ represent the initial and final velocities, respectively. By obtaining the analytical expression for a_i, it can be substituted into the trajectory, velocity, and acceleration expressions for further calculation.

$$\begin{cases} a_0 = q_0 \\ a_1 = \dot{q}_0 \\ a_2 = \dfrac{3(q_f - q_0)}{t_f - t_0} - \dfrac{(2\dot{q}_0 + \dot{q}_f)}{t_f - t_0} \\ a_3 = -\dfrac{2(q_f - q_0)}{(t_f - t_0)^3} + \dfrac{(q_0 + q_f)}{t_f - t_0} \end{cases} \tag{6}$$

3.2. Quintic polynomial interpolation function

In certain specific working scenarios, there are requirements for the robot's motion trajectory to have constraints on acceleration. In these cases, the acceleration function of the cubic polynomial can change only at a fixed rate, following a first-order form. However, the cubic polynomial may not satisfy the requirement of continuous acceleration. To address this, a higher-order quintic polynomial interpolation method can be employed for trajectory planning. The expression for quintic polynomial interpolation is defined, and by taking derivatives, expressions for trajectory position, velocity, and acceleration can be obtained.

$$\begin{cases} q(t) = a_0 + a_1(t-t_0) + a_2(t-t_0)^2 + a_3(t-t_0)^3 + a_4(t-t_0)^4 + a_5(t-t_0)^5 \\ \dot{q}(t) = a_1 + 2a_2(t-t_0) + 3a_3(t-t_0)^2 + 4a_4(t-t_0)^3 + 5a_5(t-t_0)^4 \\ \ddot{q}(t) = 2a_2 + 6a_3(t-t_0) + 12a_4(t-t_0)^2 + 20a_5(t-t_0)^3 \end{cases} \tag{7}$$

where *q(t)* represents the joint angle trajectory, and a_i *(i = 0, 1, ..., 5)* are the coefficients of the quintic polynomial. Similar to the cubic polynomial, the quintic polynomial takes into account the initial and final joint angle positions and the constraints on joint angle velocities. However, it also includes an additional equation to ensure continuous acceleration, denoted by $\ddot{q}(t_0) = \ddot{q}_0$, $\ddot{q}(t_f) = \ddot{q}_f$. By considering the constraint conditions, the expression for the polynomial coefficient parameters can be derived.

$$\begin{cases} a_0 = q_0 \\ a_1 = \dot{q}_1 \\ a_2 = \dfrac{\ddot{q}_0}{2} \\ a_3 = \dfrac{20(q_f - q_0) - (8\dot{q}_f + 12\dot{q}_0)(t_f - t_0) - 3(\ddot{q}_0 - \ddot{q}_f)(t_f - t_0)^2}{2(t_f - t_0)^3} \\ a_4 = \dfrac{-30(q_f - q_0) + (14\dot{q}_f + 16\dot{q}_0)(t_f - t_0) + (3\ddot{q}_0 - 2\ddot{q}_f)(t_f - t_0)^2}{2(t_f - t_0)^4} \\ a_5 = \dfrac{12(q_f - q_0) - 6(\dot{q}_f + \dot{q}_0)(t_f - t_0) + (\ddot{q}_0 - \ddot{q}_f)(t_f - t_0)^2}{2(t_f - t_0)^5} \end{cases} \tag{8}$$

3.3. Piecewise polynomial interpolation

In the case of multi-node trajectory interpolation, the constraints of continuous position, velocity, and acceleration can only be satisfied when at least five polynomials are used for cubic polynomial interpolation. When interpolating multi-segment trajectory points using quintic polynomials, each segment requires more constraints due to its higher order [30–33]. This results in a large amount of calculations and high time cost, making it unsuitable for tasks with fewer path points for small castings sorting robot arm. Therefore, a "3-5-3" mixed piecewise polynomial interpolation function is proposed by combining cubic and quintic polynomials. This approach not only addresses the issue of angle mutation in cubic polynomial interpolation programming but also solves the problem of excessive angular velocity in quintic polynomial interpolation programming while reducing calculation complexity and saving time. The work task of the sorting robot arm for small castings is simplified by converting it into four path points in Cartesian space. These path points are then converted into joint angles through inverse kinematics calculations (Section 2.1). The entire path is divided into three sections, each consisting of two adjacent path points. A general formula describing how joint angles vary with time under certain constraints is derived.

$$\begin{cases} q_{i1}(t) = a_{i10} + a_{i11}t + a_{i12}t^2 + a_{i13}t^3 & ,t \in (t_0, t_1) \\ q_{i2}(t) = a_{i20} + a_{i21}t + a_{i22}t^2 + a_{i23}t^3 + a_{i24}t^4 + a_{i25}t^5, t \in (t_1, t_2) \\ q_{i3}(t) = a_{i30} + a_{i31}t + a_{i32}t^2 + a_{i33}t^3 & ,t \in (t_2, t_3) \end{cases} \tag{9}$$

In this context, $q_{i1}(t)$, $q_{i2}(t)$ and $q_{i3}(t)$ represent the trajectories of the "3-5-3" segmented polynomial interpolation. The coefficients of each segment function are denoted by a_i, where $i = 0, 1, \ldots, n$ and n represent the number of joints.

$$M = \begin{bmatrix} t_1^3 & t_1^2 & t1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 3t_1^2 & 2t_1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6t_1 & 2 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_2^5 & t_2^4 & t_2^3 & t_2^2 & t_2 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 5t_2^4 & 4t_2^3 & 3t_2^2 & 2t_2 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 20t_2^3 & 12t_2^2 & 6t_2 & 2 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & & & \cdots & & & \cdots & & & 0 & t_3^3 & t_3^2 & t_3 & 1 \\ 0 & & & \cdots & & & \cdots & & & 0 & 3t_3^2 & 2t_3 & 1 & 0 \\ 0 & & & \cdots & & & \cdots & & & 0 & 6t_3 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & & & \cdots & & & \cdots & & & 0 \\ 0 & 0 & 1 & 0 & 0 & & & \cdots & & & \cdots & & & 0 \\ 0 & 1 & 0 & 0 & 0 & & & \cdots & & & \cdots & & & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{10}$$

The joint angles of the four intermediate points, as well as the velocity and acceleration of both the initial and end points, are all known to be zero. Additionally, there is continuity in velocity and acceleration between adjacent path points. To find the 14 unknown coefficients, a combination of cubic polynomials and quintic polynomials is used to derive polynomial coefficients. M represents a polynomial matrix, where t_1, t_2, t_3 denote the time used by three respective polynomials. q_0, q_1, q_2, q_3 represent trajectory joint angles for each of the four waypoints, respectively. A denotes a matrix composed of unknown parameter a_i. By multiplying matrix M with matrix Q, we obtain a matrix equation that can solve for the coefficient a_i.

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & q_3 & 0 & 0 & q_0 & 0 & 0 & q_2 & q_1 \end{bmatrix}^T \tag{11}$$

$$A = M^{-1} * Q \tag{12}$$

4. Genghis Khan shark optimizer (GKSO)

4.1. GKSO background

The Genghis Khan shark optimization algorithm is a population-based optimization algorithm

that takes inspiration from the individual and collective behaviors of Genghis Khan sharks in their oceanic habitat. It is designed to emulate their hunting, movement, foraging, and self-protective escape behaviors, which collectively form the basis of a comprehensive mathematical model for optimization. These four biological behaviors provide mathematical expressions for the four stages of the algorithm. The diagram below illustrates the four behaviors of Genghis Khan sharks [24].

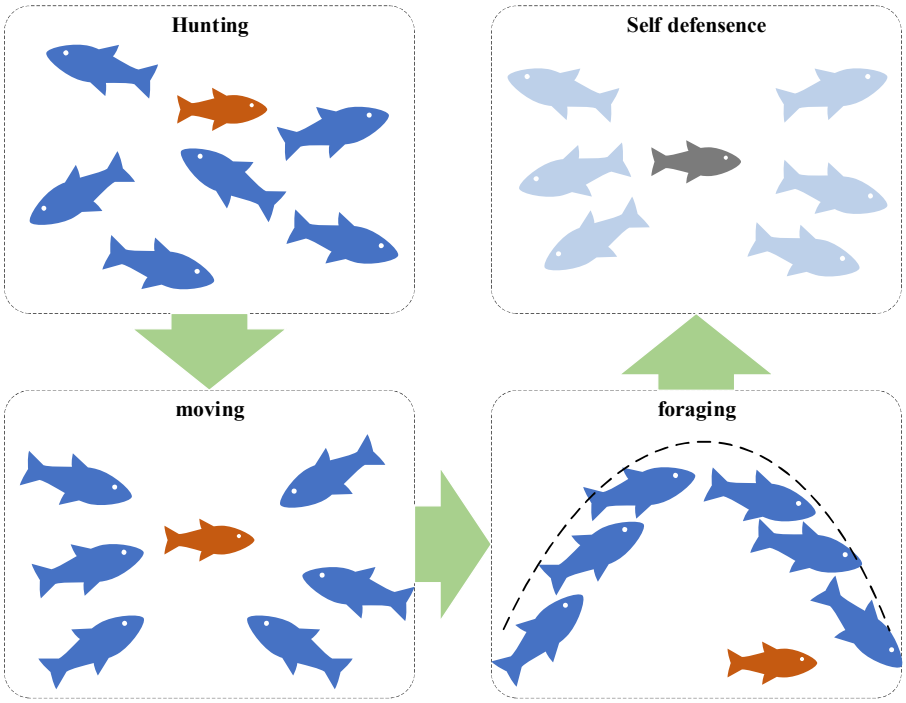


Figure 3. Diagram of the four biological behaviors of the Genghis Khan shark.

4.2. Mathematical model

4.2.1. Hunting stage

In order to ensure their safety and effectively search for food in unfamiliar marine environments, Genghis Khan sharks tend to linger near the seabed, conducting broad spatial searches. During the initial phase, the school of sharks patrols in different directions until they locate the optimal position for their target prey. This space is bounded by upper and lower limits, within which random positions are marked as optimal. The mathematical formula for initializing and updating these positions is as follows:

$$X_i^j(t+1) = X_i^j + \frac{lb_j + rand * (ub_j - lb_j)}{iter}, i = 1,2,\cdots,N, j = 1,2,\cdots,D, iter = 1,2,\cdots,T. \quad (13)$$

Where $X_i^j(t+1)$ represents the position of the i-th individual in the j-th dimension at time $t+1$, ub_j and lb_j are the upper and lower bounds in the j-th dimension, *rand* is a random number in the range of [0,1], N represents the total population size, D represents the problem dimension, *iter* denotes the current iteration number, and T is the maximum number of iterations.

4.2.2. Moving stage

After identifying the initial optimal position, Genghis Khan sharks rely on their exceptional olfactory senses to progressively close in on the prey's location. The mathematical formula for their movement towards the target position is as follows:

$$\widehat{X}_i^j(t+1) = s*(X_{best}^j(t) - X_i^j(t)) \tag{14}$$

In the equation, s represents the olfactory influence factor during the movement towards the target, X^j_{best} represents the optimal target position at time t, and s is determined by the concentration of odor emitted by the target prey. The mathematical model associated with this is as follows:

$$s = mI^r \tag{15}$$

Among these parameters, m is a non-negative constant that holds special significance as its value directly affects the overall convergence speed of the GKSO algorithm. After thorough comparison, it has been determined that setting m to 1.5 yields optimal algorithm performance. Another parameter, I, represents the attribute intensity, which is contingent upon the individual's capabilities. Additionally, r is a random number within the range of [0,1] and reflects the individual's degree of absorption towards odor information. The value of r directly influences the behavior of GKSO. When $r = 0$, it signifies that the algorithm has not detected the target prey and should continue exploring outward. Conversely, when $r = 1$, it indicates complete absorption of the odor, prompting the algorithm to move directly towards the target prey. At this point, the algorithm is likely to converge to the optimal solution. However, it is noteworthy that when $r = 1$, the value of s may also be relatively high. While this brings the algorithm closer to the optimal solution, it simultaneously reduces the overall exploration capacity. Furthermore, there is a possibility that a newly discovered optimal solution may surpass the current global optimum, limiting the algorithm's ability to further evolve. Hence, the choice of r holds significant importance.

To better approach the optimal solution, the algorithm retains the position of the previous best solution during the optimization process and updates the positions of other best solutions based on it. The position update formula is as follows:

$$X_i^j(t+1) = \frac{\widehat{X}_i^j(t+1) + X_{i-1}^j(t)}{2} \tag{16}$$

4.2.3. Foraging stage

During the foraging process, Genghis Khan sharks adopt a parabolic-shaped attack strategy for cooperative hunting, as shown in the diagram. The position update formula for this strategy is as follows:

$$X_i^j(t+1) = X_{best}^j(t) + rand*(X_{best}^j(t) - X_i^j(t)) + \lambda * p^2 * (X_{best}^j(t) - X_i^j(t)) \tag{17}$$

Among them, the value of λ is a random number that can take either 1 or -1. The parameter p controls the step size of Genghis Khan sharks during the foraging phase. A larger value of p tends to favor global exploration, while gradually reducing the value of p shifts the focus towards local

exploitation. The calculation formula for p is as follows:

$$p = 2*\left\{1-\left(\frac{t}{T}\right)^{\frac{1}{4}}+\left|\omega(t+1)\right|*[(\frac{t}{T})^{\frac{1}{4}}-(\frac{t}{T})^{3}]\right\} \tag{18}$$

The weight coefficient $|\omega(t + 1)|$ at time $t + 1$ is calculated using the following formula:

$$\left|\omega(t+1)\right| = 1-2\omega^{4}(t) \tag{19}$$

The initial value of $\omega(0)$ is typically set to 0.1.

4.2.4. Self-protection stage

During foraging, Genghis Khan sharks may encounter factors that pose a threat to their safety. In such situations, they exhibit a behavior of lightening their body color and swiftly escaping. The mathematical model for this behavior is as follows:

$$\begin{cases} X_i^j(t+1) = X_i^j(t)+k_1(z_1X_{best}^j(t)-z_2X_k^j(t))+k_2\rho(a_3(X2_i^j(t)-X1_i^j(t)))+\frac{a_2}{2}(X_{u1}^j(t)-X_{u2}^j(t)) \ \ if\ z_1<0.5 \\ X_i^j(t+1) = X_{best}^j(t)+k_1(z_1X_{best}^j(t)-z_2X_k^j(t))+k_2\rho(z_3(X2_i^j(t)-X1_i^j(t)))+\frac{z_2}{2}(X_{u1}^j(t)-X_{u2}^j(t)) \ \ \text{otherwise} \end{cases} \tag{20}$$

Among them, k_1 is a random number between -1 and 1, and k_2 follows a normal distribution with a mean of 0 and a standard deviation of 1. z_1, z_2, and z_3 are three random numbers. The calculation formula is as follows:

$$\begin{cases} z_1 = l_1*2*rand+(1-l_1) \\ z_2 = l_1*rand+(1-l_1) \\ z_3 = l_1*rand+(1-l_1) \end{cases} \tag{21}$$

The value of l_1 is a binary random number that can take either 0 or 1. ρ is an adaptive coefficient, and its calculation formula is as follows, which is related to the calculation of α and β:

$$\alpha = \left|\beta*\sin(\frac{3\pi}{2}+\sin(\frac{3\pi\beta}{2}))\right| \tag{22}$$

$$\beta = \beta_{\min}+(\beta_{\max}-\beta_{\min})*(1-(\frac{iter}{T})^{3})^{2} \tag{23}$$

$$\rho = \alpha*(2*rand-1) \tag{24}$$

β_{min} is 0.2 and β_{max} is 1. The formulas for calculating $X1_i^j$ and $X2_i^j$ in the equation are as follows:

$$X1_i^j(t) = lb_j+rand*(ub_j-lb_j) \tag{25}$$

$$X2_i^j(t) = lb_j + rand * (ub_j - lb_j) \tag{26}$$

The formula for calculating $X_k^j(t)$ is as follows:

$$X_k^j(t) = l_2 * (X_p^j(t) - X_r^j(t)) + X_r^j(t) \tag{27}$$

Among them, $X_p^j(t)$ represents the randomly selected solution at time t ($p \in 1, 2, ..., N$). Similarly, l_2 and l_1 are binary random numbers that can take either 0 or 1. $X_r^j(t)$ is a set of randomly generated initial solutions at time t. X_{u1}^j and X_{u2}^j are two randomly selected solutions. By decomposing the position update expression, we can see that the self-defense phase actually involves searching the space near the optimal solution using multiple solutions. This can be considered as a form of neighborhood search. This self-defense mechanism helps to prevent the GKSO algorithm from getting trapped in local optima and improves exploration efficiency.

4.2.5. Summary of the algorithm

The complexity of this algorithm primarily depends on the initialization of random solutions, position updates, and fitness evaluation. In GKSO, there are stages where new initialization updates occur, which enhance global search capability by introducing random solutions for position updates. The position update formula at each step is subject to upper and lower bounds, and the generation of certain solutions relies on constraints. The specific constraints are tailored to the characteristics of the function problem. In this study, we optimize and impose constraints on the position, velocity, and acceleration. The specific constraint conditions are designed based on the fitness function, but they will not be discussed in detail here.

To better illustrate the structure of the GKSO algorithm, a flowchart has been created, as shown in Figure 4.

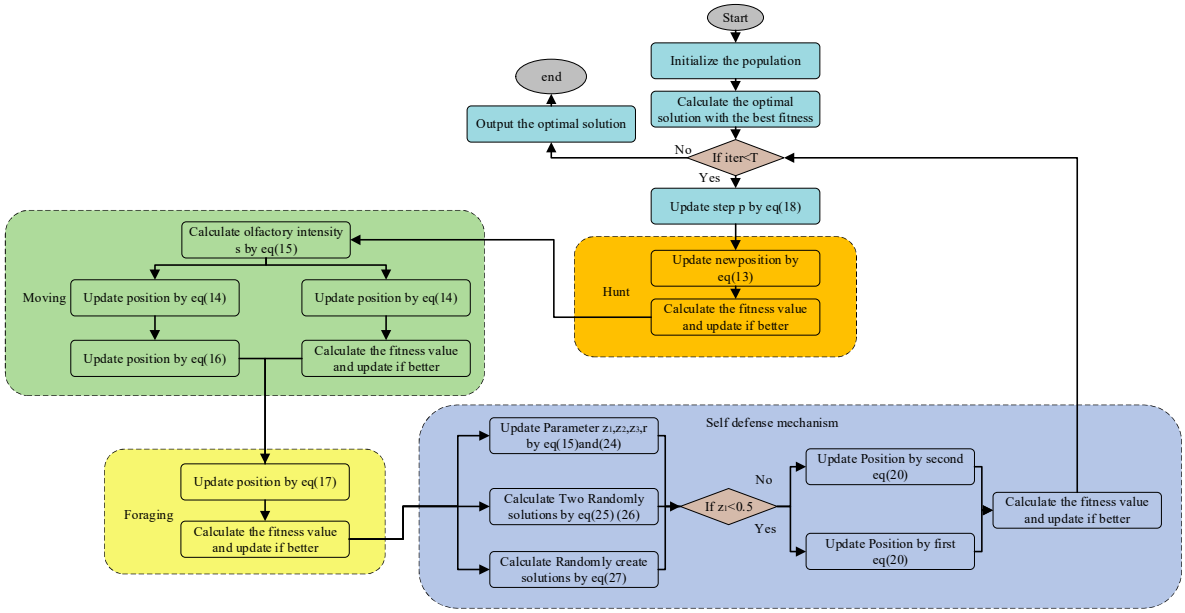


Figure 4. GKSO algorithm flow chart.

5. Experiment

5.1. Objective function

Based on Chapter 3, an interpolation function for the robotic arm's trajectory can be derived. However, in practical trajectory planning, explicit time intervals are not provided, making it challenging to determine the unknown coefficients for each interpolation segment. To address this issue, the GKSO algorithm is employed in this study to determine the optimal interpolation times that satisfy velocity and acceleration constraints.

During the trajectory optimization process, an effective approach is to directly optimize the time variable, t. This reduces the dimensionality of the objective function, significantly reducing the computational complexity of the GKSO algorithm. The fitness function and boundary constraints established for the optimization process are as follows:

$$\begin{aligned} T_{all} &= t_1 + t_2 + t_3 = \sum_{j=0}^{3} t_j \\ &\begin{cases} \left|q_{ij}(t)\right| \le Q\max, j = 1,2,3 \\ \left|\dot{q}_{ij}(t)\right| \le V\max, j = 1,2,3 \\ \left|\ddot{q}_{ij}(t)\right| \le A\max, j = 1,2,3 \end{cases} \end{aligned} \tag{28}$$

T_{all} represents the total operation time of the robotic arm, while t_1, t_2, and t_3 denote the running times of the three interpolation functions. The objective function is designed to comply with boundary constraints. By utilizing the "3-5-3" interpolation function described in Chapter 3, we derive the problem matrix and simultaneously optimize the position, velocity, and acceleration. During the experimental process, specific constraint conditions are set, and adjustments can be made to accommodate portions that exceed the boundaries in the calculation of the relevant fitness function.

In this study, our main goal is to minimize the total time, with shorter durations considered more optimal. For portions that exceed the boundaries, the GKSO algorithm replaces them with boundary values during the hunting phase. Subsequent position updates depend on the design of the fitness function. In this study, we handle portions that exceed the boundaries by replacing their fitness values with infinity, ensuring that they are not included in the point position list during subsequent updates. This approach allows for the overall optimization calculation to proceed without interference.

5.2. Work task

In order meet the requirements of a small-scale casting sorting robotic arm, specific motion trajectories need to be defined. The current robotic arm used in casting production factories primarily focuses on the task of transporting and sorting castings that have been piled up on a conveyor belt. It operates within the workspaces of machine tools, conveyor belts, and processing areas. The workspace is determined based on known joint parameters, and relative task objectives have been established, as depicted in Figure 5. The end effector is required to move sequentially through four designated waypoints, labeled A, B, C, and D, and place the items in the adjacent processing area. It then follows the same path back for sorting purposes. Point A is located on the conveyor belt. Points B and C are

located closer to the robotic arm, about 400 mm apart from each other with some height difference between the two points. Point D is located at the target position. As shown in Figure 5, the total shortest linear distance is about 2736 mm. The expected target is to make the trajectory as close to the shortest path as possible while keeping the trajectory smooth without shock or vibration during operation.

Figure 5. Work task path.

The coordinates of these four waypoints are calculated using the robot kinematics principles outlined in Chapter 2, converting them from Cartesian coordinates to joint angle space. The resulting joint angles are as follows.

Table 3. Joint angular path points.

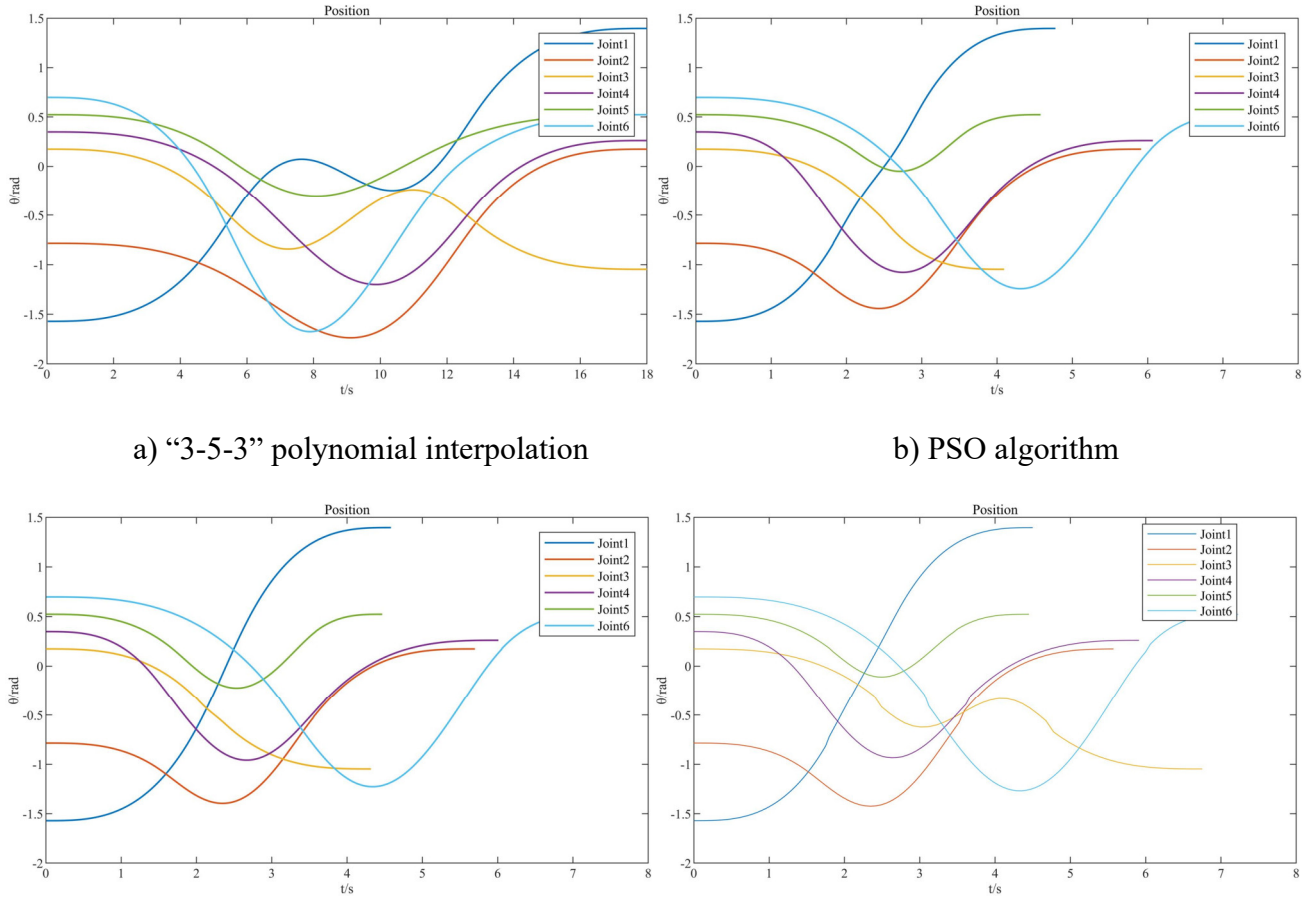
Number of joints / Number of points	1	2	3	4	5	6
A	-1.57	-0.79	0.17	0.35	0.52	0.70
B	-0.79	-1.05	-0.34	0	0.17	-0.35
C	0.61	-0.52	-0.61	-0.35	0.35°	0.17
D	1.40	0.17	-1.05	0.26	0.52	0.52

5.3. Experimental simulation

Based on the given DH parameters, a model of the casting sorting robotic arm was constructed. Four provided waypoints were used as reference points, and Table 1 specifies the angle, angular velocity, and angular acceleration constraints for each joint. To facilitate the experiment, the average values were used. Considering time optimization, a "3-5-3" polynomial interpolation with significant fluctuations in the acceleration at the transition points was selected to avoid detrimental vibrations to the robotic arm. The maximum allowable joint velocity was set at 1.5 rad/s, and the maximum allowable joint acceleration was set at 2 rad/s^2. The population size was set to 50, and the maximum number of iterations was set to 50.

Assuming initial times t_1, t_2, and t_3 are 3 seconds, 8 seconds and 3 seconds respectively, a comparative analysis was conducted between the results obtained from the "3-5-3" configuration

without PSO planning, the traditional PSO method, the improved PSO algorithm (referred to as I-PSO), and the GKSO algorithm. The PSO algorithm parameters were set as follows: the inertial influence factor was set to 0.9, and the individual and social learning factors were both set to 2. In the I-PSO algorithm, a linearly decreasing inertia weight and dynamic learning factor improvement were implemented [34–43]. Since the interpolation coefficients of the polynomial segments are time-dependent, each optimization may yield different combinations. Therefore, after conducting multiple experiments, an optimal time solution graph was obtained. The total time obtained from multiple iterations was compared in Table 4 for a comprehensive evaluation.



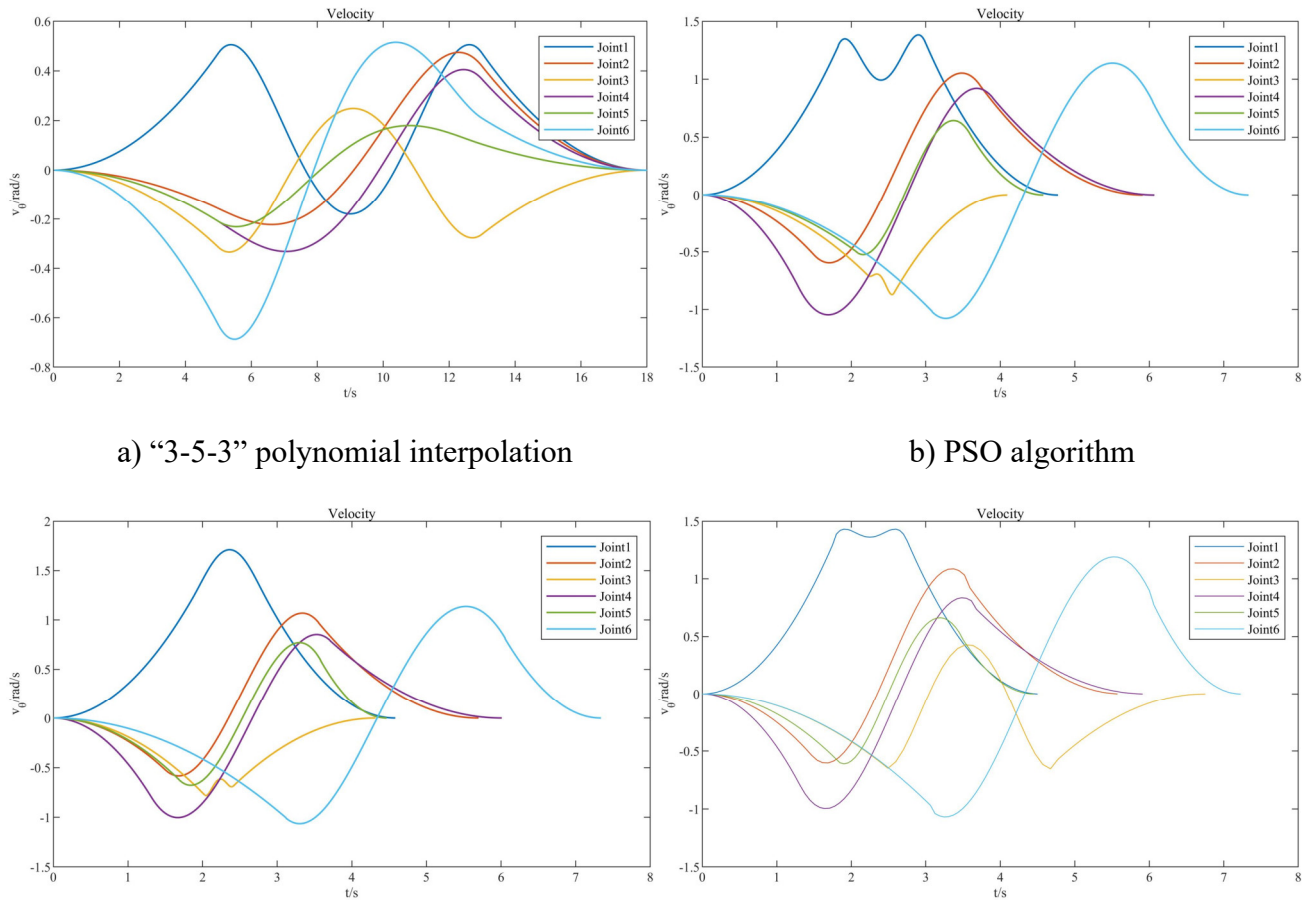
a) "3-5-3" polynomial interpolation

b) PSO algorithm

c) I-PSO algorithm

d) GKSO algorithm

Figure 6. Comparison of trajectory and time change.



a) "3-5-3" polynomial interpolation

b) PSO algorithm

c) I-PSO algorithm

d) GKSO algorithm

Figure 7. Velocity versus time change.

In Figures 6–8, a) represents trajectory planning using only the "3-5-3" segmented interpolation, b) represents the traditional PSO algorithm, c) represents the I-PSO algorithm, and d) represents the GKSO algorithm. From the trajectory, velocity, and acceleration, it is evident that the GKSO algorithm significantly shortens the time required to reach the target waypoints. The original total running time for a single joint is 18 seconds, but the optimized total running time can be reduced to a minimum of 4.109 seconds, which represents an optimization rate of approximately 77.2% compared to the unoptimized case. When compared to the PSO algorithm, the GKSO algorithm shows superior optimization results. Analyzing multiple joints, apart from joint 3 where the I-PSO algorithm outperforms, the difference between the GKSO algorithm and the I-PSO algorithm is relatively small for the other joints. However, specific numerical comparisons show that the GKSO algorithm exhibits slightly better optimization results. The trajectory and velocity obtained from the "3-5-3" interpolation planning demonstrate excellent continuity overall. Although the optimized positions show slight variations compared to the original positions, the velocities and accelerations remain within acceptable ranges, closely approaching V_{max} and a_{max}, with minimal fluctuations. The trajectory appears smooth overall, satisfying the constraints of each joint, and confirming the feasibility and effectiveness of the algorithm in enhancing the efficiency of the robotic arm.

Here is a smoothed version:

Based on comparisons of the trajectory data, in the PSO algorithm joint 3 reaches the target

position first, followed by joints 5, 1, 2, 4, and 6, with a final time of 7.45 seconds. The reaching order is the same for the I-PSO algorithm. The running time for joint 5 is optimized to approximately 4.49 seconds in the PSO algorithm and 4.48 seconds in the I-PSO algorithm. In the GKSO algorithm, the running time is around 4.44 seconds, showing little difference. The optimization of joints 3 and 6 results in more noticeable changes to running time. Joint 6 has the longest running time of 7.28 seconds. With the GKSO algorithm, joint 5 is the first to complete its motion, followed by joints 1, 2, 5, 3, and 6, with an overall running time for joint 6 of 7.10 seconds - shorter than the optimization times of the other two algorithms.

Each joint reduces a certain amount of running time. For example, joints 1, 2, 4 and 5 reach their positions ahead of the equivalent interpolation points for the other joints during runtime. After arriving at the position, they no longer change and maintain the joint angle, waiting for the next interpolation point to arrive. Finally, the remaining time is spent waiting for the last joint to complete its motion and finish rotating the joint angle.

In the velocity comparison graph, between 1.75 s and 3.00 s, joint 1 exhibits more noticeable changes. Under algorithm optimization, the difference between other joints except for joints 1 and 3 was not significant. However, the optimization time of the GKSO algorithm was slightly better. The GKSO algorithm displays smaller fluctuations and a smaller maximum amplitude in comparison to the other two algorithms. After optimization using the GKSO algorithm, the maximum speed was reached at 1.97 s and 2.97 s, approximately 1.47 rad/s. For the PSO algorithm, the maximum speed of 1.4 rad/s was reached at 2.24 s. As for the I-PSO algorithm, the maximum speeds of approximately 1.77 rad/s were reached around 1.85 s and 2.74 s. In terms of velocity fluctuation, the GKSO algorithm and the optimized PSO algorithm exhibit a bimodal pattern, while the IPSO algorithm follows a unimodal pattern. The PSO and IPSO algorithms demonstrate larger variations, whereas the GKSO algorithm shows smaller variations. The optimization objective of the employed algorithm is the total running time of each joint. Thus, while shortening the time and adhering to velocity and acceleration constraints, the inherent nature of the "3-5-3" polynomial interpolation function may lead to noticeable fluctuations at the connection points. Nevertheless, by setting lower acceleration constraint values, the vibrations caused by fluctuations in the robotic arm are reduced, and even the maximum vibration does not impede the arm's operation. To mitigate unnecessary vibrations, non-uniform B-spline optimization is applied at the connection points. As illustrated in the Figure 8, the unoptimized acceleration graph demonstrates desirable continuity, with minimal numerical fluctuations within an acceptable range. Building upon optimized time, the PSO and IPSO algorithms exhibit substantial variations in each joint, particularly joint 3, during velocity analysis. While the GKSO algorithm performs comparatively worse in terms of optimized time, it showcases superior stability and smaller fluctuation amplitudes. Considering each joint, the GKSO algorithm yields more stable acceleration variations, characterized by smooth curves devoid of inflection points. When completing the same route task, the GKSO algorithm attains shorter time durations, more stable velocities, and reduced vibrations, thereby exemplifying superior algorithmic performance. Similarly, after the trajectory is completed within the optimized runtime, the angular position is not changed. The subsequent velocity and acceleration are both zero.

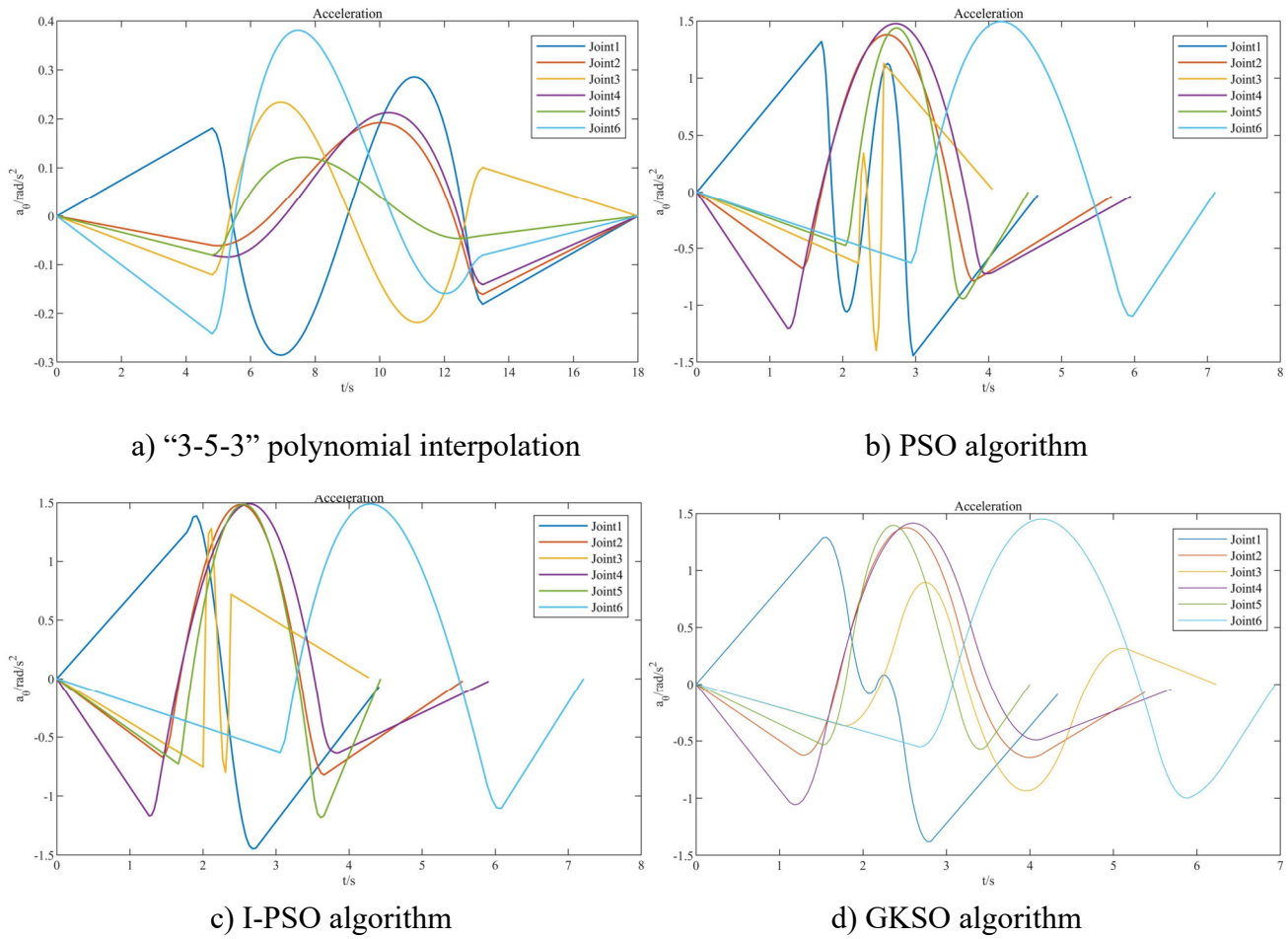


Figure 8. Acceleration and time change.

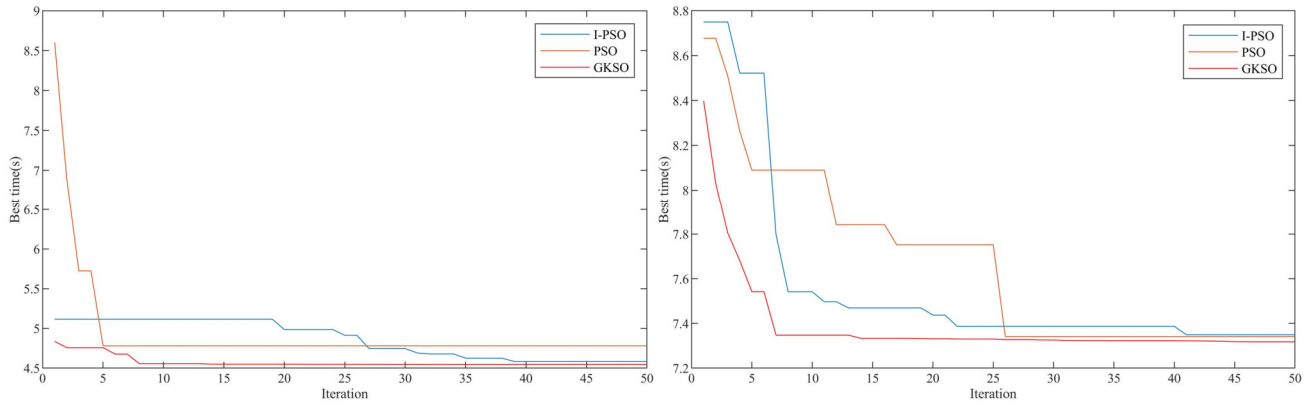


Figure 9. Variations in partial joint runtime with iteration number.

The convergence graphs in Figure 9, a) and b) represent the convergence of time and iteration for different joints. The graphs show the convergence of GKSO, traditional PSO, and I-PSO algorithms in optimizing time. From the graphs, it is evident that the GKSO algorithm converges the fastest, followed by the I-PSO algorithm. The GKSO algorithm demonstrates excellent overall convergence, almost reaching the optimal solution, indicating a significant improvement in convergence compared to the

other algorithms. Furthermore, the improved algorithm outperforms the traditional PSO algorithm by achieving smaller optimal time solutions and avoiding the trap of local optima. Moreover, the GKSO algorithm shows better optimization than the I-PSO algorithm. After 50 iterations, the convergence is already close to the optimal solution, and the GKSO algorithm converges faster. Compared to the traditional PSO and I-PSO algorithms, the GKSO algorithm achieves smaller optimal time solutions, making it a superior algorithm.

To validate the GKSO algorithm's capability in seeking initial optimal solutions, 8 groups of experiments were conducted as shown in Table 4. Under equal population sizes and iteration counts, the solutions were obtained and the times of the initial solutions from the first calculation were recorded for comparison. As shown, the GKSO algorithm generated initial calculation times that were mostly better than PSO and I-POS, with a small minority slightly lower possibly due to random numbers coincidentally being selected close to optimal. The results not only demonstrate GKSO's advantage in the early stage but also validate the algorithm's robustness.

Table 4. Comparison of initial solution computation.

Joint / Algorithm	Experiment number	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
PSO algorithm Initial time solution (s)	1	8.4098	7.1704	8.9544	8.1209	7.8547	8.8324
	2	7.9198	6.6726	8.2748	7.9653	7.7378	8.6228
	3	8.7916	7.6681	7.5922	9.0822	6.5264	9.1036
	4	8.1696	6.1719	7.6175	8.4861	7.6593	9.8066
	5	7.5208	6.9103	8.8100	7.1629	6.1390	9.7259
	6	6.9100	8.1284	8.0368	6.8938	7.0409	9.2878
	7	7.7727	7.2811	7.5904	9.0667	6.1309	10.1242
	8	7.9547	7.1014	7.3895	7.2746	6.9102	9.0220
IPSO algorithm Initial time solution (s)	1	6.5781	6.6019	8.2768	9.0012	5.8138	10.0710
	2	7.5423	6.4004	7.4877	7.1267	7.1999	8.8870
	3	6.7233	6.5709	7.9847	8.6081	5.4322	7.5025
	4	8.3360	6.2058	7.2968	8.2743	5.5754	9.3354
	5	6.2108	6.8971	7.8647	7.0906	6.0873	8.9467
	6	6.3753	7.3511	7.8584	7.5814	4.9933	8.7924
	7	8.1775	7.2881	6.4751	6.8552	5.6674	9.4112
	8	8.2808	6.6139	6.1443	7.2670	7.6661	7.9648
GKSO algorithm Initial time solution (s)	1	5.5387	6.1930	7.5979	6.3257	4.8661	7.9096
	2	5.1007	6.9222	7.2373	7.0316	4.9755	7.4689
	3	4.9898	6.4846	7.4874	6.1898	5.4697	8.0156
	4	6.1106	6.0770	7.3171	6.6270	4.8989	7.3879
	5	4.7932	7.0215	7.1610	7.5530	4.9587	7.7137
	6	4.9024	6.5184	7.1402	6.0367	5.5395	7.8925
	7	5.9273	6.6714	7.2846	7.1147	5.3979	8.0432
	8	5.2854	6.6528	7.1431	6.8349	4.7269	7.8857

It is challenging to discern the time differences between the traditional PSO algorithm and the improved algorithm from the trajectory, velocity, and acceleration graphs. To compare the differences

between the two algorithms, the initial times for the cubic polynomial interpolation are adjusted to 5 s, while the initial times for the quintic polynomial interpolation are set to 8 s. Table 5 provides a separate breakdown of the running times for each joint to facilitate comparison. Additionally, multiple experiments are conducted to compare the results for different iteration numbers and population sizes. Based on multiple comparisons, the GKSO algorithm and the I-PSO algorithm show minimal differences. The optimal solutions for joints 1, 2, 4, and 6 differ by no more than 0.6 s. However, overall, the GKSO algorithm achieves slightly better optimization objectives. For joint 3, the GKSO algorithm successfully avoid local optima through its defensive measures, resulting in an optimal time solution with minimal fluctuations and demonstrating good performance in terms of acceleration. Furthermore, as the iteration numbers and population sizes increase, the optimal solutions show a decreasing trend, further enhancing the effectiveness of the optimization algorithm.

Table 5. Iterative optimization time comparison.

Number of joint / Condition	Algorithm	1	2	3	4	5	6
Population size 50, iterations 50	Basical PSO (time/s)	4.72	5.91	5.10	6.39	4.70	7.35
	Improved PSO (time/s)	4.61	5.73	4.75	6.21	4.42	7.38
	GKSO (time/s)	4.53	5.68	5.81	6.01	4.41	7.29
Population size 50, iterations 100	Basical PSO (time/s)	4.69	5.79	5.20	6.31	4.83	7.31
	Improved PSO (time/s)	4.39	5.71	4.74	5.98	4.36	7.23
	GKSO (time/s)	4.33	5.70	5.42	5.72	4.22	7.16
Population size 70, iterations 50	Basical PSO (time/s)	4.65	5.81	5.06	6.29	4.71	7.29
	Improved PSO (time/s)	4.31	5.66	4.75	6.03	4.43	7.25
	GKSO (time/s)	4.27	5.32	5.41	5.98	4.39	7.11

5.4. Experiment

To validate the practical applicability of the optimization algorithm in real-world scenarios for the robotic arm, the algorithm's pre- and post-optimized trajectory points are implemented in the casting sorting robotic arm. The trajectory plot depicted in the figure illustrates the simulated trajectories in MATLAB before and after optimization using the GKSO and PSO algorithms. The blue line represents the trajectory obtained from the GKSO algorithm, the black line represents the trajectory obtained from the I-PSO algorithm, and the red line represents the trajectory obtained from the traditional PSO optimization.

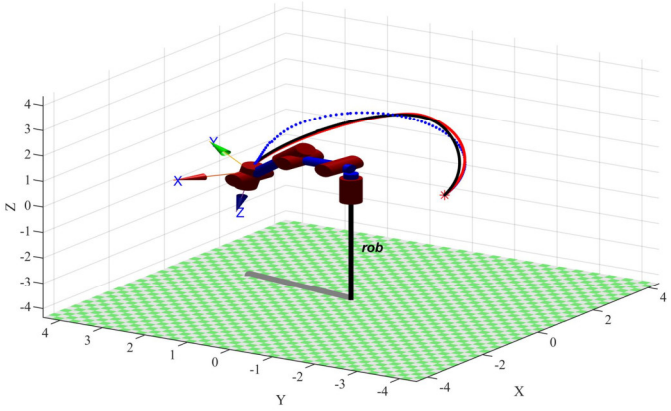


Figure 10. Trajectory simulation after algorithm optimization.

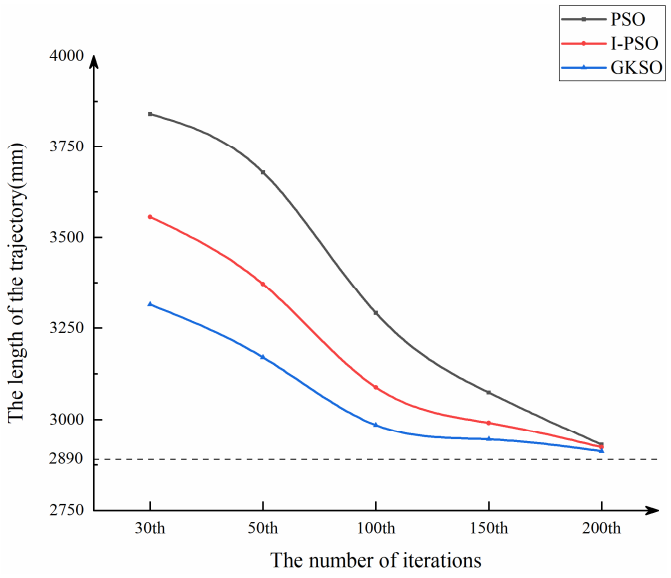


Figure 11. Trajectory length comparison data graph.

It is evident that, in the case of the I-PSO algorithm and PSO optimization, the middle section of the trajectory is longer. The trajectory lines are divided into multiple points and fed into the robotic arm for actual operation. The trajectory lengths optimized by different algorithms through 30, 50, 100, 150, and 200 iterations of optimized runtime were respectively calculated, as shown in Figure 11. All algorithms exhibited rapid convergence behavior with increasing iterations. Among them, the initial solution of the GKSO algorithm was better in the first 150 iterations, with convergence closer to the optimal solution. After 200 iterations, there was little difference in the overall optimization effect of the algorithms on this robotic arm trajectory planning problem. At this point, they all approached the shortest trajectory length, which to some extent validated the certain advantages of the GKSO algorithm in trajectory planning. To account for the challenge of real-time adjustment of acceleration in the control segment, linear averaging is applied between each acceleration and deceleration segment. Three experiments were conducted without specifying interpolation time, prioritizing safety by setting

the extremum at 30% of the robotic arm's maximum speed and acceleration. The trajectories obtained after 30, 60, and 100 iterations are imported into the robotic arm for sorting, as shown in Figure 12. The results are documented in Table 6. The overall operation unfolds smoothly without significant fluctuations, and there are no instances of jamming at the transition points of the trajectory, which further solidifies the reliability of the desired trajectory.

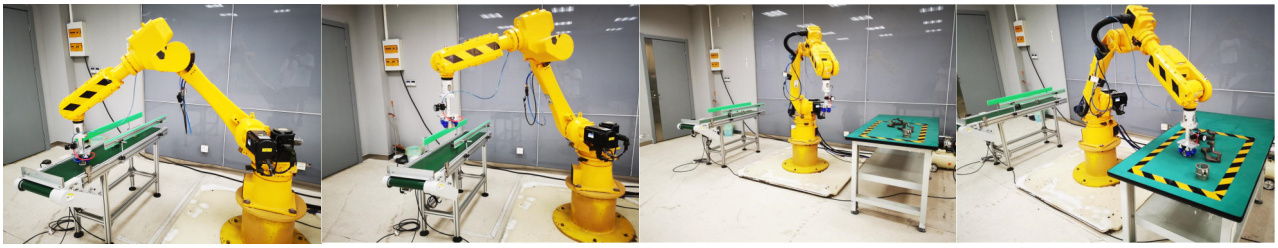


Figure 12. Experiment of robotic arm trajectory operation.

Table 6. Trajectory running time comparison.

Iterations	30	60	100
GKSO	39.87 s	36.57 s	33.05 s
Improved PSO	43.08 s	38.85 s	33.59 s
PSO	54.72 s	48.89 s	38.46 s

In order to consider the runtime of the algorithm itself, a timing function is implemented to measure the computation time for obtaining the optimal solution. With 50 iterations and a population size of 50, multiple optimizations are performed, and the average runtime is calculated as shown in the Table 7. The overall difference in runtime is within 2 seconds. Furthermore, after incorporating dynamic learning factors and adaptive inertia weights, the difference in runtime for the I-PSO algorithm is within 1 second. Although there is a slight disparity, the time saved from multiple repetitions of optimization can offset this relatively small difference in industrial production settings.

It is important to note that we did not focus on optimizing and simplifying the GKSO algorithm itself, resulting in the use of code that may not be the most concise. Additionally, in cases where multiple nested formulas are present, parallel computing cannot be utilized to reduce runtime.

Table 7. The running time of the algorithm for each joint.

Algorithm / Number of joint	GKSO algorithm runtime (s)	PSO algorithm runtime (s)	I-PSO algorithm runtime (s)
Joint 1	7.939962	5.004269	6.093648
Joint 2	7.777234	5.16282	6.37203
Joint 3	7.768724	5.100063	6.552508
Joint 4	7.774511	6.36556	6.872196
Joint 5	7.958547	5.187522	6.685268
Joint 6	8.028946	6.087946	6.443644

6. Conclusions

We present a time-optimized trajectory planning method for the casting sorting robotic arm using the GKSO (Genghis Khan Shark Optimization) algorithm. We utilize the Monte Carlo method to compute the robotic arm's workspace and employ a "3-5-3" segmented polynomial interpolation technique for end-effector trajectory planning. Furthermore, we compare and analyze the simulated position, velocity, and acceleration profiles of the trajectories optimized using the GKSO algorithm, the traditional PSO (Particle Swarm Optimization) algorithm, and the I-PSO (Improved Particle Swarm Optimization) algorithm. The results confirm that the GKSO algorithm outperforms the other two algorithms, yielding the best time optimization results.

Moreover, the GKSO algorithm exhibits remarkable continuity in the optimized trajectories, adhering to the robotic arm's kinematic constraints. Compared to the other algorithms, the GKSO algorithm demonstrates lower fluctuations, higher stability, and enhanced reliability. Finally, when the calculated trajectory points are implemented in the actual operation of the robotic arm, minimal fluctuations are observed. In conclusion, qw successfully achieve the objective of optimizing the time-optimal trajectories for the casting sorting robotic arm.

In terms of future research directions, there are several potential areas to explore. First, as the algorithm is in the early stages of research, there may be a tendency for the generation of initial solutions to be concentrated, thereby limiting the algorithm's global search capability in the initial phase. To address this, one possible direction is to consider the integration of chaotic mapping techniques. This could help improve the generation of initial solutions and enhance the algorithm's ability to explore the global solution space.

Second, further studies can be conducted to investigate and optimize the parameters associated with the four stages of the algorithm. By refining and fine-tuning these parameters, it is possible to improve the overall performance and efficiency of the algorithm.

Additionally, there is research value in exploring strategies to optimize and simplify the algorithm itself. By streamlining the algorithm, it can become more efficient and easier to implement in practical applications.

Furthermore, in real-world industrial production scenarios, there may be situations that involve multiple points and requirements beyond the scope of this study. Therefore, it is worth exploring how the algorithm can be further optimized to handle such multi-point scenarios and provide accurate and effective solutions.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This paper was supported by Anhui Province Graduate Education Quality Project (2022cxcysj107) and Natural Science Foundation of Anhui Province: 2208085ME128.

Conflict of interest

The authors declare that there are no conflicts of interest.

References

1. Y. Chen, L. Li, Collision-free trajectory planning for dual-robot systems using B-splines, *Int. J. Adv. Rob. Syst.*, **14** (2017). https://doi.org/10.1177/1729881417728021
2. R. Marco, C. Fabio, S. Marco, A. Alessandra, A new framework for joint trajectory planning based on time-parameterized B-splines, *Comput.-Aided Des.*, **154** (2023), 103421. https://doi.org/10.1016/j.cad.2022.103421
3. Y. Li, H. Tian, D. G. Chetwynd, An approach for smooth trajectory planning of high-speed pick-and-place parallel robots using quintic B-splines, *Mech. Mach. Theory*, **126** (2018), 479–490. https://doi.org/10.1016/j.mechmachtheory.2018.04.026
4. H. Wang, W. Heng, J. Huang, B. Zhao, L. Quan, Smooth point-to-point trajectory planning for industrial robots with kinematical constraints based on high-order polynomial curve, *Mech. Mach. Theory*, **139** (2019), 284–293. https://doi.org/10.1016/j.mechmachtheory.2019.05.002
5. H. Wang, Q. Zhao, H. Li, R. Zhao, Polynomial-based smooth trajectory planning for fruit-picking robot manipulator, *Inf. Process. Agric.*, **9** (2022), 112–122. https://doi.org/10.1016/j.inpa.2021.08.001
6. X. Li, H. Lv, D. Zeng, Q. Zhang, An improved multi-objective trajectory planning algorithm for kiwifruit harvesting manipulator, *IEEE Access*, **11** (2023), 65689–65699. https://doi.org/10.1109/ACCESS.2023.3289207
7. Ü. Dinçer, M. Çevik, Improved trajectory planning of an industrial parallel mechanism by a composite polynomial consisting of Bézier curves and cubic polynomials, *Mech. Mach. Theory*, **132** (2019), 248–263. https://doi.org/10.1016/j.mechmachtheory.2018.11.009
8. F. Lin, L. Shen, C. Yuan, Z. Mi, Certified space curve fitting and trajectory planning for CNC machining with cubic B-splines, *Comput.-Aided Des.*, **106** (2019), 13–29. https://doi.org/10.1016/j.cad.2018.08.001
9. S. Lu, B. Ding, Y. Li, Minimum-jerk trajectory planning pertaining to a translational 3-degree-of-freedom parallel manipulator through piecewise quintic polynomials interpolation, *Adv. Mech. Eng.*, **12** (2020). https://doi.org/10.1177/1687814020913667
10. X. Zhao, M. Wang, N. Liu, Y. Tang, Trajectory planning for 6-DOF robotic arm based on quintic polynormial, in *Proceedings of the 2017 2nd International Conference on Control, Automation and Artificial Intelligence (CAAI 2017)*, 2017. https://doi.org/10.2991/CAAI-17.2017.23
11. G. Wu, S. Zhang, Real-time jerk-minimization trajectory planning of robotic arm based on polynomial curve optimization, *Proc. Inst. Mech. Eng., Part C: J. Mech.*, **236** (2022), 10852–10864. https://doi.org/10.1177/09544062221106632
12. M. Dupac, Smooth trajectory generation for rotating extensible manipulators, *Math. Methods Appl. Sci.*, **41** (2018), 2281–2286. https://doi.org/10.1002/mma.4210
13. P. Boscariol, D. Richiedei, Energy-efficient design of multipoint trajectories for Cartesian robots, *Int. J. Adv. Manuf. Technol.*, **102** (2019), 1853–1870. https://doi.org/10.1007/s00170-018-03234-4
14. A. E. Ezugwu, A. K. Shukla, R. Nath, A. A. Akinyelu, J. O. Agushaka, H. Chiroma, Metaheuristics: a comprehensive overview and classification along with bibliometric analysis, *Artif. Intell. Rev.*, **54** (2021), 4237–4316. https://doi.org/10.1007/s10462-020-09952-0
15. J. Zhang, Q. Meng, X. Feng, H. Shen, A 6-DOF robot-time optimal trajectory planning based on an improved genetic algorithm, *Rob. Biomimetics*, **5** (2018), 3. https://doi.org/10.1186/s40638-018-0085-7

16. K. Shi, Z. Wu, B. Jiang, H. R. Karimi, Dynamic path planning of mobile robot based on improved simulated annealing algorithm, *J. Franklin Inst.*, **360** (2023), 4378–4398. https://doi.org/10.1016/j.jfranklin.2023.01.033
17. X. Zhang, F. Xiao, X. Tong, J. Yun, Y. Liu, Y. Sun, et al., Time optimal trajectory planing based on improved sparrow search algorithm, *Front. Bioeng. Biotechnol.*, **10** (2022), 852408. https://doi.org/10.3389/fbioe.2022.852408
18. T. Wang, Z. Xin, H. Miao, H. Zhang, Z. Chen, Y. Du, Optimal trajectory planning of grinding robot based on improved whale optimization algorithm, *Math. Probl. Eng.*, **2020** (2020), 3424313. https://doi.org/10.1155/2020/3424313
19. I. Carvajal, E. A. Martínez-García, R. Lavrenov, E. Magid, Robot arm planning and control by τ-Jerk theory and vision-based recurrent ANN observer, in *2021 International Siberian Conference on Control and Communications (SIBCON)*, (2021), 1–6. https://doi.org/10.1109/SIBCON50419.2021.9438857
20. E. Özge, A. Bekir, Trajectory planning for a 6-axis robotic arm with particle swarm optimization algorithm, *Eng. Appl. Artif. Intell.*, **122** (2023), 106099. https://doi.org/10.1016/j.engappai.2023.106099
21. G. Chen, W. Peng, Z. Wang, J. Tu, H. Hu, D. Wang, et al., Modeling of swimming posture dynamics for a beaver-like robot, *Ocean Eng.*, **279** (2023), 114550. https://doi.org/10.1016/j.oceaneng.2023.114550
22. G. Chen, Y. Xu, C. Yang, X. Yang, H. Hu, X. Chai, et al., Design and control of a novel bionic mantis shrimp robot, *IEEE/ASME Trans. Mechatron.*, **28** (2023), 3376–3385. https://doi.org/10.1109/TMECH.2023.3266778
23. K. Wu, L. Chen, K. Wang, M. Wu, W. Pedrycz, K. Hirota, Robotic arm trajectory generation based on emotion and kinematic feature, in *2022 International Power Electronics Conference (IPEC-Himeji 2022-ECCE Asia)*, (2022), 1332–1336. https://doi.org/10.23919/IPEC-Himeji2022-ECCE53331.2022.9807205
24. G. Hu, Y. Guo, G. Wei, L. Abualigah, Genghis Khan shark optimizer: a novel nature-inspired algorithm for engineering optimization, *Adv. Eng. Inf.*, **58** (2023), 102210. https://doi.org/10.1016/j.aei.2023.102210
25. R. V. Ram, P. M. Pathak, S. J. Junco, Inverse kinematics of mobile manipulator using bidirectional particle swarm optimization by manipulator decoupling, *Mech. Mach. Theory*, **131** (2019), 385–405. https://doi.org/10.1016/j.mechmachtheory.2018.09.022
26. P. Golla, S. Ramesh, S. Bandyopadhyay, Kinematics of the Hybrid 6-Axis (H6A) manipulator, *Robotica*, **41** (2023), 2251–2282. https://doi.org/10.1017/S0263574723000334
27. A. V. Antonov, A. S. Fomin, Inverse kinematics of a 5-DOF hybrid manipulator, *Autom. Remote Control*, **84** (2023), 281–293. https://doi.org/10.1134/S0005117923030037
28. J. Q. Gan, E. Oyama, E. Rosales, H. Hu, A complete analytical solution to the inverse kinematics of the Pioneer 2 robotic arm, *Robotica*, **23** (2005), 123–129. https://doi.org/10.1017/S0263574704000529
29. G. Zhong, B. Peng, W. Dou, Kinematics analysis and trajectory planning of a continuum manipulator, *Int. J. Mech. Sci.*, **222** (2022), 107206. https://doi.org/10.1016/j.ijmecsci.2022.107206
30. C. Wang, F. Ding, L. Ling, S. Li, Design of a teat cup attachment robot for automatic milking systems, *Agriculture*, **13** (2023), 1273. https://doi.org/10.3390/agriculture13061273

31. A. Messaoudi, R. Sadaka, H. Sadok, Matrix recursive polynomial interpolation algorithm: An algorithm for computing the interpolation polynomials, *J. Comput. Appl. Math.*, **373** (2020), 112471. https://doi.org/10.1016/j.cam.2019.112471
32. M. Ivan, V. Neagos, A representation of the interpolation polynomial, *Numerical Algorithms*, **88** (2021), 1215–1231. https://doi.org/10.1007/s11075-021-01072-2
33. X. Liu, G. Lin, W. Wei, Adaptive transition gait planning of snake robot based on polynomial interpolation method, *Actuators*, **11** (2022), 222. https://doi.org/10.3390/act11080222
34. A. Shrivastava, V. K. Dalla, Multi-segment trajectory tracking of the redundant space robot for smooth motion planning based on interpolation of linear polynomials with parabolic blend, *Proc. Inst. Mech. Eng., Part C: J. Mech.*, **236** (2022), 9255–9269. https://doi.org/10.1177/09544062221088723
35. D. Wang, D. Tan, L. Liu, Particle swarm optimization algorithm: an overview, *Soft Comput.*, **22** (2017), 387–408. https://doi.org/10.1007/s00500-016-2474-6
36. V. Trivedi, P. Varshney, M. Ramteke, A simplified multi-objective particle swarm optimization algorithm, *Swarm Intell.*, **14** (2020), 83–116. https://doi.org/10.1007/s11721-019-00170-1
37. Y. Zhang, X. Liu, F. Bao, J. Chi, C. Zhang, P. Liu, Particle swarm optimization with adaptive learning strategy, *Knowledge-Based Syst.*, **196** (2020), 105789. https://doi.org/10.1016/j.knosys.2020.105789
38. A. G. Gad, Particle swarm optimization algorithm and its applications: a systematic review, *Arch. Comput. Methods Eng.*, **29** (2022), 2531–2561. https://doi.org/10.1007/s11831-021-09694-4
39. J. Zheng, Y. Gao, H. Zhang, Y. Lei, J. Zhang, OTSU multi-threshold image segmentation based on improved particle swarm algorithm, *Appl. Sci.*, **12** (2022), 11514. https://doi.org/10.3390/app122211514
40. L. Yu, Y. Han, L. Mu, Improved quantum evolutionary particle swarm optimization for band selection of hyperspectral image, *Remote Sens. Lett.*, **11** (2020), 866–875. https://doi.org/10.1080/2150704X.2020.1782501
41. S. Obukhov, A. Ibrahim, A. A. Z. Diab, A. S. Al-Sumaitim, R. Aboelsaud, Optimal performance of dynamic particle swarm optimization based maximum power trackers for stand-alone PV system under partial shading conditions, *IEEE Access*, **8** (2020), 20770–20785. https://doi.org/10.1109/ACCESS.2020.2966430
42. X. Li, B. Tian, S. Hou, X. Li, Y. Li, C. Liu, et al., Path planning for mount robot based on improved particle swarm optimization algorithm, *Electronics*, **12** (2023), 3289. https://doi.org/10.3390/electronics12153289
43. P. Qu, F. Du, Improved particle swarm optimization for laser cutting path planning, *IEEE Access*, **11** (2023), 4574–4588. https://doi.org/10.1109/ACCESS.2023.3236006

AIMS Press