Rev. Roum. Sci. Techn.– Électrotechn. et Énerg.
Vol. **70**, *1*, pp. 109–114, Bucarest, 2025

A VELOCITY SELF-LEARNING ALGORITHM FOR TIME-OPTIMAL TRAJECTORY PLANNING ALONG THE FULLY SPECIFIED PATH – PART I

WENHU NAN[1], HAOJUN QIN[1]

Keywords: Industrial handling robot; Velocity self-learning; Hermite interpolation; Correction trajectory; Actual joint torque.

In response to the problem of uncertainty in the system dynamics model during the time-optimal trajectory planning for the industrial handling robots, a novel online self-learning model-free time-optimal trajectory planning method is proposed. First, offline kinematic constraints and the Hermite interpolation algorithm are used to obtain the optimal spline velocity curve under kinematic constraints. Then, online trajectory data of the robot's operation is collected, and the trajectory generation method using a self-learning strategy is employed to refine the trajectory iteratively, resulting in a time-optimal trajectory under actual dynamic constraints. Finally, taking the ur5e cooperative robot and ABB IRB9670-235 industrial robot as the experimental platform, experiments verify the effectiveness and efficiency of the proposed method.

1. INTRODUCTION

Time-optimal trajectory planning along fully specified paths has been widely used in industrial handling and other fields, such as the feeding and unloading operations in the manufacturing process of large parts. The theory of time-optimal trajectory planning under geometric constraints is essentially solved [1]. However, the algorithm efficiency and adaptability under dynamic constraints [2] still need to increase. The typical time-optimal trajectory planning method includes the dynamics and model-free models. At present, there are two methods based on the dynamics model: the indirect method and the direct method. The model-free method mainly includes the reinforcement learning method.

The first dynamic model method is an indirect method mainly consisting of the Pontryagin maximum principle and Dynamic programming [3–6]. The time optimal trajectory solved using the Pontryagin maximum principle is considered a “bang-bang” trajectory type and can be calculated by successive integration of the maximum and minimum acceleration. Theoretically, this approach is the quickest algorithm, as it adopts the bang-bang structure of the optimization algorithm. However, this method is difficult to apply due to the programming difficulties and the robustness issues associated with the dynamic singularities [7]. The other dynamic model method is a direct method, which is convex optimization [8–10] and types of swarm intelligence algorithms [11,12]. Convex optimization methods discretize the s-axis into segments and convert the original problem into a convex optimization problem with variables, equality, and inequality constraints. The swarm intelligence algorithm translates the time-optimal trajectory planning problem into a multi-variation optimization problem. This is like convex optimization methods in that both can consider more general constraints and objective functions, such as energy or torque rate, leading to less aggressive actuator use. The direct method has the limitations of producing suboptimal results and has low computation efficiency. Steinhauser et al. [13] present a two-step iterative learning algorithm that compensates for such model-plant mismatch and finds the time-optimal motion, improving tracking performance and ensuring feasibility. Due to an efficient solution to the path tracking problem using a sequential convex log barrier method, the delay between consecutive task executions is eliminated. Although the model error can be reduced somewhat, the joint motor's torque overload problem cannot be fundamentally solved by relying on the dynamic model.

Although the dynamic model method has made significant progress in theoretical planning speed and success rate, the actual calculation results are affected by the unmodeled part of robot system dynamics during actual operation. Therefore, the uncertainty of the dynamic model makes it challenging to apply the model-based method to the industrial handling robot in practice.

The model-free method can avoid the problem of uncertainty in a dynamic model. Yu L et al. [14] introduced the motor dynamic load constraint into the reinforcement learning environment. They modified the strategy trajectory, finally obtaining the time-optimal trajectory that meets the motor dynamic load constraint. However, reinforcement learning mainly focuses on learning in task space [15], and the efficiency of reinforcement learning is low [16]. Chatzilygeroudis K et al. [17,18] challenged microdata and proposed a data efficiency reinforcement learning model. Reinforcement learning currently requires tens of thousands of training, and the learning efficiency of trajectory planning is low. Therefore, it is a massive challenge for industrial robots to be handled in practice.

Because the dynamic model is uncertain and reinforcement learning is inefficient in current research, this paper proposes an efficient online self-learning time-optimal trajectory planning method. The parameterized trajectory is modified by learning the measured torque during the robot operation to ensure that the time-optimal speed trajectory meets the actual dynamic constraints.

The rest of the paper is organized as follows: section 2 presents the time-optimal trajectory planning problem. Section 3 introduces the online velocity self-learning algorithm. In section 4, the proposed method's performance is tested on an example of a real robot. Finally, section 5 concludes this article and discusses future research directions.

2. PROBLEM STATEMENT

The general time-optimal trajectory planning problem involves finding the optimal trajectory that can accurately follow a predefined path in the shortest possible time while adhering to various constraints. To mathematically formulate this problem, we first need to consider the dynamic model for a general robotic manipulator, consider the dynamic model for a general robotic manipulator, which can be expressed as follows,

$$\boldsymbol{\tau} = \boldsymbol{\psi}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\boldsymbol{\varphi} + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}), \tag{1}$$

[1] Key Laboratory of Heavy-Duty Flexible Robot in Mechanical Industry, Lanzhou University of Technology, Lanzhou 730050, China.
E-mails: nanwenhu@163.com, 20160028@lut.edu.cn

DOI: 10.59277/RRST-EE.2025.1.19

where $\boldsymbol{\varphi}$ represents the set of inertia parameters and $\mathbf{F}$ represents the dissipative force vector. $\boldsymbol{\tau}$ denotes the vector of generalized forces, and $\boldsymbol{\psi}$ represents the inertia coefficient matrix, which is dependent solely on the joint angle and its derivative. The inertia parameters in $\boldsymbol{\varphi}$ encompass both the inertia parameters of the robot's connecting links and the load inertia parameters.

When tracking a fixed path, for a single parameter "s" in eq. (1) that increases monotonically along the path, the result is,

$$\boldsymbol{\tau}(s) = \boldsymbol{\psi}(\mathbf{q}(s), \dot{\mathbf{q}}(s), \ddot{\mathbf{q}}(s))\boldsymbol{\varphi} + \mathbf{F}(\mathbf{q}(s), \dot{\mathbf{q}}(s)), \quad (2)$$

$$\underline{\boldsymbol{\tau}} \leqq \boldsymbol{\tau}(s) \leqq \boldsymbol{\tau}. \quad (3)$$

By analyzing Fig. 1, it can be observed that in the process of grasping, the effective end link inertia parameters of the robot are the combination of the robot end link and load G. Because the load centroid position and mass are unknown, the effective inertia parameters of the end link are unknown. Even if the parameter identification method [16] is adopted, four parameters of mass and centroid in the load inertia parameters can be obtained, but there are ten complete load inertia parameters. Moreover, different load objects need to be identified, which is time-consuming. Considering the joint friction, nonlinear gap and other factors in the process of robot operation, the actual dynamic model exhibits significant uncertainty. Based on such fact, this paper presents a novel self-learning method, where the robot utilizes this method to adapt the changing load and achieve time optimal trajectory.

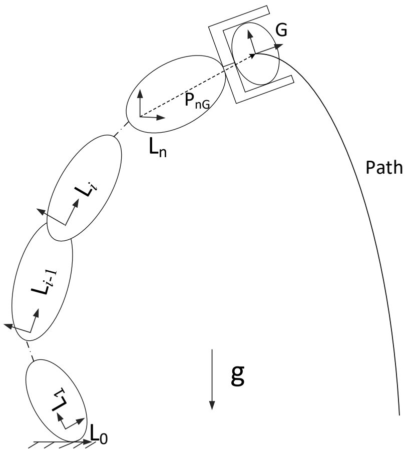

Fig. 1 – A schematic of a working robot on path constraints.

Kinematic constraints typically involve maximum joint velocities and accelerations, and is defined as,

$$\underline{\dot{\mathbf{q}}} \leqq \dot{\mathbf{q}} \leqq \overline{\dot{\mathbf{q}}} \qquad \dot{\mathbf{q}}(s) = \mathbf{q}'(s)\dot{s}, \quad (4)$$

$$\underline{\ddot{\mathbf{q}}} \leqq \ddot{\mathbf{q}} \leqq \overline{\ddot{\mathbf{q}}} \qquad \ddot{\mathbf{q}}(s) = \mathbf{q}'(s)\ddot{s} + \mathbf{q}''(s)\dot{s}^2. \quad (5)$$

Finally, the TOTP problem can be formulated mathematically as follows,

$$\min(T) \quad T = \int_0^{S_{end}} \frac{\mathrm{d}s}{\dot{s}}, \quad (6)$$

$$\text{subject to} \begin{cases} \frac{\mathrm{d}\boldsymbol{q}(s)}{\mathrm{d}t} \in [\underline{\dot{\mathbf{q}}}, \overline{\dot{\mathbf{q}}}], \\ \frac{\mathrm{d}^2\mathbf{q}(s)}{\mathrm{d}t^2} \in [\underline{\ddot{\mathbf{q}}}, \overline{\ddot{\mathbf{q}}}], \\ \boldsymbol{\tau}(s) \in [\underline{\boldsymbol{\tau}}, \overline{\boldsymbol{\tau}}], \end{cases} \quad (7)$$

where T is the total time to complete the trajectory, $s(t)$ is the time function of parameter s (to be determined), $\mathrm{d}\mathbf{q}(s)/\mathrm{d}t$ and $\mathrm{d}^2\mathbf{q}(s)/\mathrm{d}t^2$ correspond to (4) and (5) defined previously, which are kinematic constraints. The kinematic parameters of the robot can be obtained by consulting the factory parameters of the robot or by identifying the kinematic parameters. When the kinematic parameters are obtained, the MVC (maximum velocity curve) can be solved out at the path point s with (4), if all first-order constraints have been satisfied when following the formation of the MVC. However, second-order kinematic constraints only establish an upper limit for $\dot{s}$, following the formation of the MVC. Acceleration switching points are then identified, which are defined as locations in $(s, \dot{s})$ where there is a change in the constraint that governs the time optimal solution. Switching points can be identified by using direct methods [1]. Once the switching points are known, a forward integration of the second-order constraint equations is performed at the start of the path and at each switching point. The optimal path speed of the kinematic model can be obtained by referring to the method proposed by Barnett E. [2]. To achieve optimal dynamic speed, the uncertainty of the model speed needs to be addressed through interactive learning and descent methods.

The joint velocity and acceleration constraints in eq. (3) can be addressed using the method proposed by Barnett E., known as MVCK (maximum velocity curve of kinematics), which solves for the optimal velocity curve under kinematic constraints based on the kinematic parameters. However, to tackle the uncertainty of joint torque parameters in eq. (3) and (7), this paper proposes an online self-learning algorithm within the framework of traditional reinforcement learning.

3. THE ONLINE VELOCITY SELF-LEARNING ALGORITHM

Here we present a novel time-optimal trajectory planning approach called velocity self-learning, which eliminates the need for integration and the identification of acceleration switching points. The technique is inspired by the observation that the optimal speed curve can be adjusted by decreasing the MVCK along path sections where violation of constraints happens. Velocity self-learning performs a single backward speed adjustment, to create a boundary curve in $s - \dot{s}$. This simplified approach is made possible through the use of the velocity self-learning algorithm, along with actual feedback torque for determining whether the torque exceeds the limit.

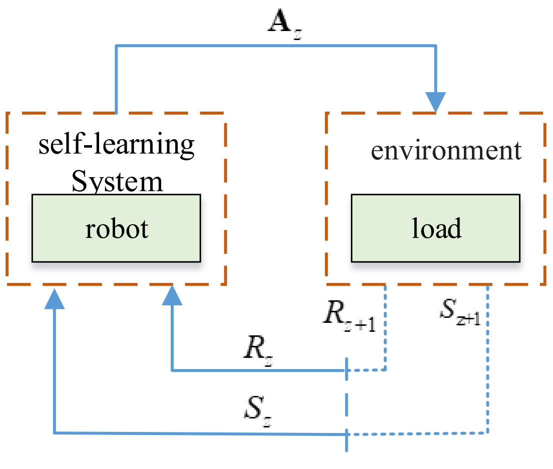

Fig. 2 – Self-learning diagram.

Reinforcement learning is a kind of self-learning system and its four basic elements are: state, action, strategy and reward. The principle is shown in Fig. 2. It is mainly to learn in the way of off-line or on-line by repeated experiment and maximize the reward through a limited number of actions to determine the best action strategy.

Traditional reinforcement learning includes **Q**-matrix learning and DQN, but **Q**-matrix learning encounters the challenge of dimensional disaster. DQN (Deep Q-Network) requires tens of thousands of iterations for learning, making it apparent that the handling robot is unsuitable for undergoing such extensive repetitive training.

Aiming at the problem of learning too many times in traditional reinforcement learning, this paper proposes the online self-learning algorithm shown in eq. (8). In essence, the online self-learning algorithm is not to adjust the network parameters or **Q** matrix values for learning, but to generate the optimal action strategy by directly adjusting the parameterized trajectory of the robot path.

$$A_{z+1} = L(S_z, R_z) \;\; z \in [0, Z_{max}]. \tag{8}$$

In eq. (8), L represents the velocity self-learning trajectory decision algorithm, which will be introduced in section 3.2. Z_{max} represents the maximum learning times. The state variables and return values are used as the input variables of the second self-learning algorithm, the output of function L is the action of the velocity self-learning algorithm. The robot generates a new action and interacts with the environment, and then continuously modifies the trajectory through the self-learning trajectory decision algorithm to obtain the time optimal trajectory that meets the requirements of actual dynamics constraints.

3.1 INTERPOLATION OPTIMIZATION FOR MVCK

An optimal interpolation technique is needed for processing input and output data for the velocity self-learning algorithm, but more importantly it is needed because interpolants will be repeatedly evaluated during backward adjustment passes. As with other dynamic programming approaches, velocity self-learning algorithm begins by representing all relevant path variables as a function of a single parameter $\boldsymbol{s}$. Although the TCP length is the most typical choice for $\boldsymbol{s}$, in many cases it may not be the most appropriate option. For a general manipulator, a better selection for s can be determined by simply integrate [2],

$$\mathrm{d}(s) = \|\mathrm{d}(\mathbf{q})\|. \tag{9}$$

This choice guarantees that all relevant joints and Cartesian variables depend on $\boldsymbol{s}$. When evaluating an interpolant for a particular value of $\boldsymbol{s}$, the optimization problem will be transformed into a single parameter variable. When the state $\boldsymbol{S}_z$ is that the robot runs a fixed path, the joint velocity of the robot,

$$\mathbf{S}_z = \{\dot{s}_{zk}, k \in [1, m]\}. \tag{10}$$

Although offline kinematic planning can achieve a kinematically time-optimal velocity curve, commonly labeled as MVCK for short, it often lacks smoothness and leads to significant vibrations when directly applied to robot control. Therefore, Hermite cubic spline interpolation method is used to interpolate the path speed, and the path velocity is as follows,

$$\dot{s} = \mathbf{V}(s). \tag{11}$$

When interpolating $\boldsymbol{V}_k$, firstly, the $\boldsymbol{V}_k(s)$ is discretized according to the fixed path length, and the path interpolation nodes meet the following requirements,

$$\Delta s = s_1 - s_0 = s_2 - s_1 = \cdots = s_c - s_{c-1}. \tag{12}$$

Let the first derivative of cubic interpolation spline function $\boldsymbol{V}(s)$ at the node s_i be $\boldsymbol{V}'(s_i) = \boldsymbol{m}_i (i = 0,1,2,\cdots c)$. To meet the splicing condition, the parameter $t = s - s_i/\Delta s$ is introduced, and the $V(s)$ expression on s_i, s_i,

$$\mathbf{V}(s) = (1 - 3t^2 + 2t^3)\mathbf{V}(s_i) + (3t^2 - 2t^3)\mathbf{V}(s_{i+1}) + (t - 2t^2 + t^3)\mathbf{m}_i + (t^3 - t^2)\mathbf{m}_{i+1}, \tag{13}$$

In eq. (13), the interpolation node of m_i is unknown. Calculating the second derivative of (13), list the strictly diagonally dominant matrix according to the equality of the left and right limits at the path node and the boundary conditions, and solve m_i by the pursuit method, to obtain the expression of the spline function $\boldsymbol{V}(s)$,

$$\mathbf{V}(s) = \begin{cases} \mathbf{V}_1(s - s_0), \; s \in [s_0, s_1], \\ \mathbf{V}_2(s - s_1), \; s \in [s_1, s_2], \\ \mathbf{V}_c(s - s_{c-1}), \; s \in [s_{c-1}, s_c]. \end{cases} \tag{14}$$

After calculating the joint path velocity interpolation function $\mathbf{V}(s)$, the derivative $\mathbf{V}(s)$ is obtained after joint path interpolation

$$\ddot{s} = \mathbf{V}'(s) \cdot \mathbf{V}(s). \tag{15}$$

When the robot is running, interpolation limiting algorithm is adopted for ensuring that the interpolated velocity trajectory does not exceed the kinematic optimal velocity curve $\boldsymbol{V}_k$. As shown in Table 1, the velocity fine-tuning parameter λ is used to locally adjust the kinematic optimal velocity curve $\boldsymbol{V}_k$. Finally, the interpolated velocity trajectory curve $\boldsymbol{V}_s$ satisfying the kinematic constraints is obtained.

Table 1
Interpolation limiting algorithm.

```
Input: $\mathbf{V}_k$
Output: $\mathbf{V}$
1.Choose node$(s_i, \mathbf{V}_k(s_i)), i = 1,2, \dots$, c, generating spline $\mathbf{V}$
2.do
3.      for (i=1:c-1)
4.            if (Segment of I spline speed exceeds its limit)
5.                  $V_k(s_i) \leftarrow V_k(s_i) - \lambda$
6.      generating new spline $V$
7.while ($V \in V_k$)
8.Save $V$ and $V_k(s_i)$
```

To get action $\mathbf{A}_z$ in (14), we substitute $\mathbf{V}_s$ into eq. (4), resulting in $\dot{\mathbf{q}}_{zk}$, then we define the action value of self-learning,

$$\mathbf{A}_z = \{\mathbf{q}_{zk}, \dot{\mathbf{q}}_{zk}, k \in [1, m]\}. \tag{16}$$

When the robot tracks the fixed path, the initial action is calculated from the kinematic interpolation velocity curve $\mathbf{V}_s$. In the subsequent learning process, it continues iterative learning to generate new state trajectory $\mathbf{S}_{z+1}$, and then obtains a new action $\mathbf{A}_{z+1}$ through formulas (5) and (21). In the z-th track operation cycle, collect the dynamic track data set, which is defined as $\boldsymbol{D}_z$,

$$\mathbf{D}_z = \{\mathbf{q}'_{zk}, \boldsymbol{\tau}'_{zk}\}, k \in [1, m]. \tag{17}$$

$\mathbf{D}_z$ includes the feedback position of the joint and the measured torque information of the joint. After the robot runs according to the predetermined trajectory every time, calculating the torque constraint state vector λ_{zk} collected for the z-th sampling according to the $\boldsymbol{D}_z$,

$$\lambda_{zk} = \begin{cases} 0 & \tau_k \in [-\tau_{kmax}, \tau_{kmax}] \\ 1 & \tau_k \notin [-\tau_{kmax}, \tau_{kmax}] \end{cases} \quad k \in [1, m] \quad (18)$$

Defining the return value of the *z*-th sampling,

$$\mathbf{R}_z = \sum_{k=1}^{m} \lambda_{zk}. \quad (19)$$

It can be seen that $\mathbf{R}_z$ is the sum of λ_{zk}. After several times of learning, if the return value $\mathbf{R}_z$ of the trajectory is zero, it means that self-learning is successful.

3.2 THE SELF-LEARNING TRAJECTORY DECISION ALGORITHM

After the state parameterization operation in section 3.1, the self-learning trajectory decision operation is carried out. The specific implementation steps are as follows:

Step 1: Calculating the return value $\mathbf{R}_z$, then, collecting the dynamic data set $\mathbf{D}_z$ and calculating the return value $\mathbf{R}_z$ according to eq. (18) and (19).

Step 2: Correcting velocity operation of path node under torque constraint. Judging return value $\mathbf{R}_z$. If it is zero, it means that learning is successful; If it is not zero, judging whether there is a torque constraint on each interpolation spline from the endpoint of the trajectory in the reverse order of the path. If it is found that the joint torque exceeds the limit at the path point *k* on spline curve segment *I*, adjusting the speed according to eq. (20),

$$\begin{cases} \dot{s}_i \geq \dot{s}_{i-1} \rightarrow \dot{s}_i = \dot{s}_i - d \\ \dot{s}_i < \dot{s}_{i-1} \rightarrow \dot{s}_{i-1} = \dot{s}_{i-1} - d \end{cases} \quad i \in [c, c-1, \cdots, 1]. \quad (20)$$

In (20), *d* is the torque correction coefficient. The velocity of each torque constraint point is corrected based on the reverse order of path points according to the eq. (20). This signifies the completion of the correction operation for the torque constraint.

Step 3: Correcting speed operation of path node under acceleration constraint. After step 2 is completed, the torque of the track is learned, but the acceleration of the track may be constrained. Using eq. (15) and (5) to calculate the joint acceleration and judging whether there is acceleration constraint on each spline curve from the termination track point in reverse order of the path. If the acceleration exceeds the limit at the path point on spline curve segment *i*, the speed is adjusted according to eq. (21),

$$\begin{cases} \dot{s}_c \geq \dot{s}_{c-1} \rightarrow \dot{s}_c = \dot{s}_c - \delta, \\ \dot{s}_c < \dot{s}_{c-1} \rightarrow \dot{s}_{c-1} = \dot{s}_{c-1} - \delta, \end{cases} \quad k \in [c, c-1, \cdots, 1]. \quad (21)$$

In eq. (21), δ is the acceleration correction coefficient. According to the reverse order of path points, from spline segment *c* to 1, the velocity of the acceleration constraint point is corrected according to eq. (21), which means that the correction operation of acceleration constraint is completed.

Step 4: According to the new spline node $[\dot{s}_0, \dot{s}_1, \cdots \dot{s}_c]$, the action A_z is calculated by (16) and (4), which is the new driving trajectory calculated by the self-learning strategy. So far, a self-learning trajectory decision algorithm is completed.

After the above four steps were completed, the time-optimal trajectory determined by actual dynamics was obtained. The following will introduce the interactive process of online self-learning.

3.3 THE INTERACTIVE PROCESS OF ONLINE SELF-LEARNING

Figure 3 shows the online interactive self-learning process is mainly composed of kinematics offline planning, environmental system and self-learning system. The interactive learning process is composed of trajectory information generated in the robot operation process and generated in the robot operation process and driving trajectory generated by the self-learning system. The interactive learning process is composed of trajectory data information collection, self-learning trajectory decision and robot operation. The specific steps are shown in Fig. 3.

Step 1: Obtaining the initial value of the driving trajectory. Firstly, the kinematics off-line planning algorithm in [2] is used to obtain $\mathbf{V}_k$, and then section 3.1 states trajectory parameter algorithm is used to parameterize $\mathbf{V}_k$, and limit the amplitude to smooth the parameterized optimal velocity trajectory to obtain the initial action value A_0, then drives the robot to track the fixed path motion, collect the motion data, and then proceed to step 2.

Step 2: Obtaining the velocity self-learning optimal trajectory decision. The trajectory decision algorithm in Section 3.2 is used for self-learning. After learning, the robot controller sends it to the driver to drive the robot and collects the dynamic data to set $\mathbf{D}_z$ again.

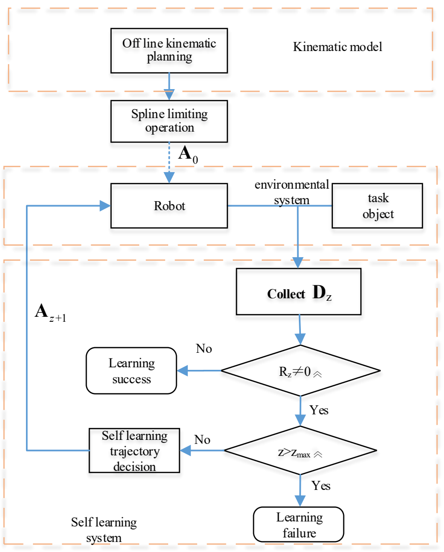

Fig. 3 – Velocity Self-learning framework.

Step 3: Detecting termination conditions. If $\mathbf{R}_z$ is not zero within the specified learning times, carrying out the self-learning optimal trajectory decision in step 2; If $\mathbf{R}_z$ is zero, it means that learning is successful; If the number of learning times *Z* exceeds Z_{max}, the algorithm fails.

4. **EXPERIMENT STUDY**

4.1 KINEMATIC OFF-LINE PLANNING INTERPOLATION OPTIMIZATION VERIFICATION

In order to verify the effectiveness and feasibility of the proposed self-learning algorithm for kinematics off-line planning, matlab2021a software on a computer with CPU frequency of 3.19G Hz and memory of 8GB is used for kinematic off-line planning research. UR5E cooperative robot is selected as the experimental platform (Fig.8). The robot kinematics parameters are shown in Table 2. Based on the robot parameters, the optimal trajectory planning research for kinematics off-line time is carried out.

Table 2

D-H parameters of UR5E robot.

i	l_i/m	α_i /(°)	d_i/m	θ_i	Max speed(°/s)
1	0	90	0	θ_1	180
2	-0.425	0	0	θ_2	180
3	-0.39225	0	0	θ_3	180
4	0	90	0.10915	θ_4	180
5	0	-90	0.09465	θ_5	180
6	0	0	0.0823	θ_6	180

To verify the effectiveness of the algorithm proposed in this paper, considering the curvature variability of the path in the actual handling operation, this paper uses the NURBS curve interpolator to generate a space star curve track with variable curvature. Among the parameters of the space star curve, the degree k = 2, the node vector $\mathbf{u}$ = [0 0 0 0 1/9 2/9 3/9 4/9 5/9 6/9 7/9 8/9 1 1 1 1 1 1 1 1], and the vertex parameters are shown in Table 3. The path generated by interpolation is shown by the black solid line in Fig. 4, and the robot TCP takes the star path shown by the black solid line. Experiments verify the effectiveness of the proposed algorithm for arbitrary path time optimal planning.

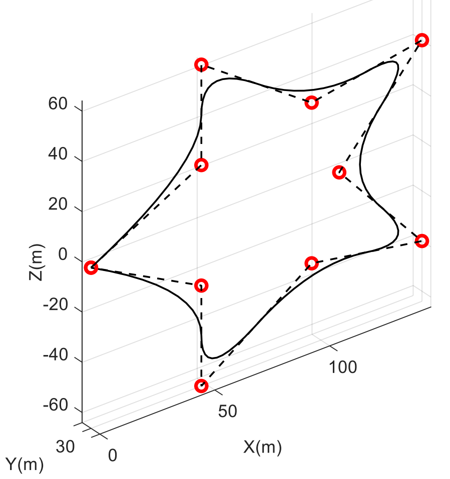

Fig. 4 – Working space diagram of star shaped handling path.

Table 3

Vertex parameters of star curve

Node serial number	Control point coordinates /m	Weight factor
1	(-0.10,0.25,0.30)	1
2	(-0.004,0.25,0.348)	1
3	(-0.004,0.25,0.428)	1
4	(0.092,0.25, 0.364)	1
5	(0.188,0.25, 0.38)	0.7
6	(0.116,0.25, 0.3)	1
7	(0.188,0.25, 0.22)	0.7
8	(0.092,0.25, 0. 236)	1
9	(-0.004,0.25,0. 172)	1
10	(-0.004,0.25,0. 252)	1

The online velocity self-learning algorithm optimizes the star trajectory, and the results are shown in Fig. 5-7. Figure 5 shows the joint path velocity on the phase plane. It can be seen that the path speed curve optimized by the online self-learning algorithm does not exceed the maximum speed curve of joint speed constraints.

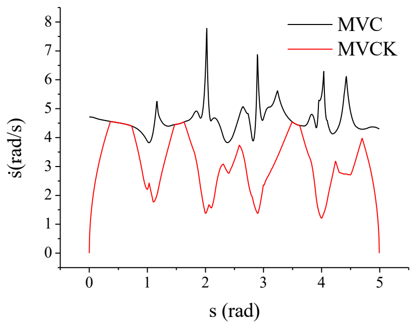

Fig. 5 – Phase plane velocity diagram.

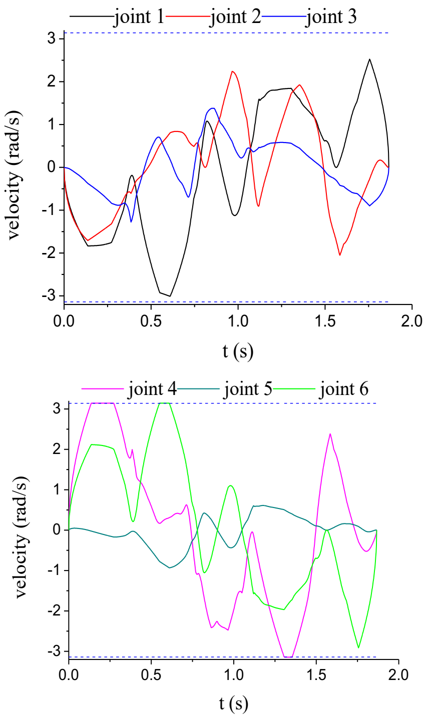

Fig. 6 – Joint velocity diagram.

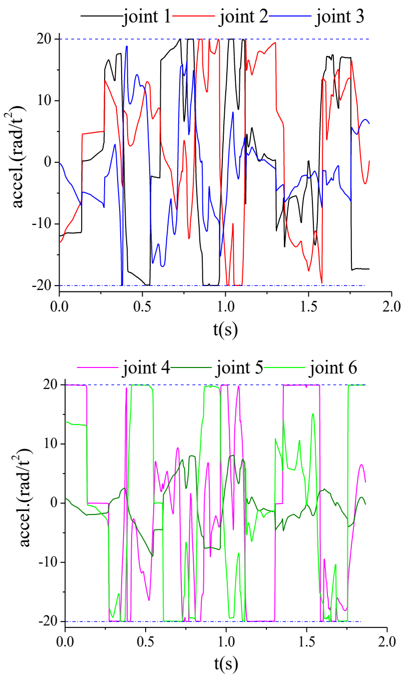

Fig. 7 – Joint acceleration diagram.

Figures 6 and 7 show the joint velocity and acceleration curves optimized by the online self-learning algorithm, respectively. The joint velocity and acceleration curves optimized by the offline kinematics algorithm are within the constraints, and the results optimized by the offline kinematics algorithm meet the kinematic constraints, which verifies the effectiveness of the proposed algorithm.

Received on 25, December, 2023

REFERENCES

1. H. Pham, Q.C. Pham, *A new approach to time-optimal path parameterization based on reachability analysis*, IEEE Transactions on Robotics, **34**, *3*, pp. 645–659 (2018).
2. E. Barnett, C. Gosselin, *A bisection algorithm for time-optimal trajectory planning along fully specified paths*, IEEE Transactions on Robotics, **37**, *1*, pp. 131–145 (2020).
3. K. Shin, N. Mckay, *Minimum-time control of robotic manipulators with geometric path constraints*, IEEE Transactions on Automatic Control, **30**, *6*, pp. 531–541 (1985).
4. J.A. Rojas-Quintero, F. Dubois, H.C. Ramírez-De-Ávila*, Riemannian formulation of Pontryagin's maximum principle for the optimal control of robotic manipulators*, Mathematics, **10,** *7*, pp. 1117–1128 (2022).
5. K. Shin, N. McKay, *A dynamic programming approach to trajectory planning of robotic manipulators*, IEEE Transactions on Automatic Control, **31**, *6*, pp. 491–500 (1986).
6. S. Singh, M. Leu, *Optimal trajectory generation for robotic manipulators using dynamic programming*, J. Dyn. Syst. Meas. Control, **109**, *2*, pp. 88–96 (1987).
7. Q.C. Pham, *A general, fast, and robust implementation of the time-optimal path parameterization algorithm*, IEEE Transactions on Robotics, **30**, *6*, pp. 1533–1540 (2014).
8. D. Verscheure, B. Demeulenaere, J. Swevers et al., *Time-optimal path tracking for robots: A convex optimization approach*, IEEE Transactions on Automatic Control, **54**, *10*, pp. 2318–2327 (2009).
9. Z. Kingston, M. Moll, L.E. Kavraki, *Sampling-based methods for motion planning with constraints*, Annual review of control, robotics, and autonomous systems, **1**, *1*, pp. 159–185 (2018).
10. F. Debrouwere, W.V. Loock, G. Pipeleers et al., *Time-optimal path following for robots with convex–concave constraints using sequential convex programming*, IEEE Transactions on Robotics, **29**, *6*, pp. 1485–1495 (2013).
11. A. Tharwat, M. Elhoseny, A.E. Hassanien et al., *Intelligent Bézier curve-based path planning model using chaotic particle swarm optimization algorithm*, Cluster Computing, **22**, *4*, pp. 1–22(2019).
12. L. Zhang, Y. Wang, X. Zaho et al., *Time-optimal trajectory planning of serial manipulator based on adaptive cuckoo search algorithm*, J. of Mechanical Science and Technology, **35**, *7*, pp. 3171–3181 (2021).
13. A. Steinhauser, J. Swevers, *An efficient iterative learning approach to time-optimal path tracking for industrial robots*, IEEE Transactions on Industrial Informatics, **14**, *11*, pp. 5200–5207 (2018).
14. L. Yu, X. Shao, Y. Wei, K. Zhou, *Intelligent land-vehicle model transfer trajectory planning method based on deep reinforcement learning*, Sensors, **18**, *9*, pp. 2905 (2018).
15. X. Lei, Z. Zhang, P. Dong, *Dynamic path planning of unknown environment based on deep reinforcement learning*, J. of Robotics (2018).
16. S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, *Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection*, The International Journal of Robotics Research, **37**, *4-5*, pp. 421–436 (2018).
17. K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, J.B. Mouret, *Black-box data-efficient policy search for robotics*, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp. 51–58 (2017).
18. S.A. Khader, H. Yin, P. Falco et al., *Data-efficient model learning and prediction for contact-rich manipulation tasks*, IEEE Robotics and Automation Letters, **5**, *3*, pp. 4321–4328 (2020).