2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)
June 28-30, 2023. Seattle, Washington, USA

Motion Profile Optimization in Industrial Robots using Reinforcement Learning

Yunshi Wen, Honglu He, Agung Julius, John T. Wen

***Abstract*—Path tracking problems are challenging with the absence of dynamic models and information about robot controllers. This paper presents a method of optimizing a motion profile constructed using a set of pre-defined motion primitives and a speed command to track a spatial trajectory with high accuracy, speed, and uniform motion using industrial robots. We use a bi-level optimization approach that optimizes execution accuracy using reinforcement learning and execution speed using bi-section search. We train and evaluate the reinforcement learning policy in simulation for an ABB robot. Experiment results demonstrate that the learned policy reduces the optimization cost to achieve the desired specifications. Additionally, the trained policy can generalize to trajectories not included in the training set.**

***Index Terms*—Reinforcement Learning, Industrial Robot, Motion Primitive, Trajectory Optimization, Trajectory Tracking**

I. Introduction

Many applications of industrial robots involve trajectory-tracking tasks, including cold spraying [1], welding [2], surface grinding [3], etc. In such tasks, a robot's tool center point (TCP) needs to track complex geometric trajectories with high accuracy, high speed, and uniform motion. However, in virtually all industrial robots, the users do not have access to low-level control functions (e.g., at the joint-torques level). Instead, the most common method to program industrial robots is to convert the desired TCP trajectory into a *motion profile*. A motion profile consists a set of waypoints connected by segments of *motion primitives* (e.g., straight line motions). The robot's controller executes this sequence of motion segments to produce the actual TCP trajectory, typically with some tracking errors (see Sec. II for more details).

In many applications, the system performance is specified by two factors: trajectory tracking accuracy (i.e., the tracking errors should be small) and trajectory tracking speed (i.e., the trajectory traversal speed should be high and uniform). Therefore, the conversion from the desired TCP trajectory to the motion profile is typically done to optimize these two (often conflicting) performance factors.

Existing works in trajectory optimization for industrial robots focus on one of the factors in trajectory-tracking tasks, including TCP position [4], speed [5], energy [6], force [7], etc. Most of the methods use a feedback controller design explicitly for only one of the factors. Current approaches in minimum-time path tracking usually require the dynamic model of the robot [8], [9] or assume the accurate executions given the constraints [10]. However, in many applications of industrial robots, the robot is a "black-box" model where the user gives commands and observes the execution trajectory that may deviate from the expected trajectory. Current practice in the industry of tuning the execution error is mostly manual and time-consuming.

The authors are with the Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute, Troy, NY, USA. Emails: {weny2,heh6,juliua2,wenj} @rpi.edu.

There are many ways the conversion from the desired TCP trajectory to the motion profile can be optimized ([11]–[13]). Typically, such a process would involve the following steps [14]:

1) **Redundancy Resolution**: In some tasks, after the TCP position and orientation are specified, there are still redundant degrees of freedom (DoF). E.g., the task may only specify that the TCP should be normal to a given surface, leaving one DoF unspecified. Further, there may be additional translational and/or rotational DoF for the surface above. The `Redundancy Resolution` step optimizes the free DoF according to an objective related to some trajectory properties. The product of this step is a fully resolved robot trajectory, either in the joint space or in TCP position and orientation space.
2) **Motion Profile Optimization**: The robot trajectory produced by the `Redundancy Resolution` step is converted into the motion profile by optimizing the sequence of motion primitives and their parameters (see details in Sec. III).

Both steps involve non-convex and combinatorial optimization, typically executed in time-consuming iterative algorithms. If the trajectories are constantly changing, e.g. in additive manufacturing, speeding up this conversion process is crucial.

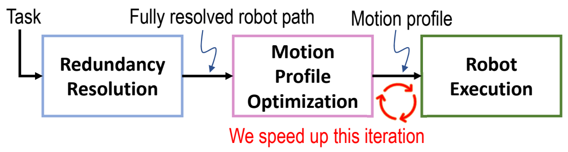
Task
Fully resolved robot path
Motion profile
Redundancy Resolution
Motion Profile Optimization
Robot Execution
We speed up this iteration

Fig. 1: Workflow of Motion Planning using Motion Primitives

In this paper, we assume the `Redundancy Resolution` step is complete and focus on speeding up the `Motion Profile Optimization` step. While both steps are challenging and necessary, speeding up the `Motion Profile Optimization` step is more critical for the following reason. Since the steps are executed in series, we can first use (approximate) models or simulations (e.g., Robot-Studio for ABB robots [15]) of the robot to execute the `Redundancy Resolution` step. However, the models/simulations are never exact. Therefore, in the `Motion Profile Optimization` step we need to run iterations on the actual robot (see Fig. 1). Our proposed algorithm is able to reduce the number of iterations needed in this process, and thereby reducing the time needed for the conversion process.

Our algorithm speeds up the waypoint adjustment process in the `Motion Profile Optimization` step. Waypoint adjustment is an optimization process where the robot iteratively executes the motion profile and adjusts its waypoints to improve the tracking accuracy. To reduce the number of iterations needed, we propose a method to train a policy for adjusting waypoints in a motion profile using *Reinforcement Learning* (RL) [16]. Note that the use of data-driven optimization techniques (RL or others) is necessitated by the lack of explicit model that can map the waypoints to the actual robot trajectory with sufficient accuracy, as demanded by the applications.

The main contributions of our approach in this paper are as follows:

- We propose a motion profile optimization method to optimize both execution errors and speed for tracking spatial trajectories using industrial robots.
- We formulate the iterative waypoint adjustment process as a reinforcement learning problem that does not require explicit dynamic models and information about the robot controller.
- We train a well-performing policy using existing RL algorithms. On average, our policy achieves 17.7x speed-up in time compared to the conventional gradient-descent-based method.
- We demonstrate the generality of our proposed approach, both in simulations and experiments, in executing trajectories not included in the RL training dataset.

The remainder of this paper is organized as follows. In Sec. II, we present notations and backgrounds of the trajectory-tracking problem, motion primitives executions, and specifications. In Sec. III, we present the optimization problem formulation and methods to solve it. In Sec. IV, we discuss the formulation and procedure of our proposed RL method. We demonstrate the performance of our method from experiment results in Sec. V. We conclude and discuss future works in Sec. VI.

II. Preliminaries

A. Notations

We use the symbol $\mathbf{p} \in \mathbb{R}^3$ to denote the TCP position. The symbol $\boldsymbol{\beta} \in \mathbb{R}^3$ denotes the TCP orientation in axis angle representation [17]. The symbol $\mathbf{q} \in \mathbb{R}^6$ denotes the joint angles in a 6-DoF industrial robot. We use the index i to refer to the i^{th} waypoint, index j to refer to the j^{th} point in a trajectory, and index k to refer to the k^{th} step in an RL episode. Other notations that we will use are summarized in Table I.

Fig. 2 shows an example of a trajectory-tracking problem. The sequence of points $\{[\mathbf{p}^j_{\text{track}}, \boldsymbol{\beta}^j_{\text{track}}]\}_{j=1}^n$ in Fig. 2(a) specifies Cartesian-space TCP position and orientation of the trajectory to track. The set of waypoints $\{[\mathbf{p}^i_{\text{wp}}, \boldsymbol{\beta}^i_{\text{wp}}]\}_{i=1}^m$ connected by motion primitives in Fig. 2(b) represents the motion profile.

TABLE I: Table of Notations

Symbol	Meaning
$\mathbf{p}^j$	TCP position of the j^{th} point
$\boldsymbol{\beta}^j$	TCP orientation of the j^{th} point
λ^j	TCP speed of the j^{th} point
$\mathbf{q}^j$	Joint angles of the j^{th} point
$\mathbf{T} = \{[\mathbf{p}^j, \boldsymbol{\beta}^j]\}_{j=1}^n$	A trajectory with n points
$\bar{\mathbf{T}}$	A normalized trajectory
$\mathbf{T}_{\text{track}}$	Trajectory to track
$\mathbf{T}_{\text{execute}}$	Actual robot execution trajectory
ψ^i	Index of the i^{th} waypoint in a trajectory
$\mathbf{a}^i$	Action (adjustment) for the i^{th} waypoint
$\mathbf{e}^i$	Positional error vector at the i^{th} waypoint
$\mathbf{T}^i_{\text{error}}$	Local error trajectory around the i^{th} waypoint
$\mathbf{T}^i_{\text{target}}$	Local target trajectory around the i^{th} waypoint
$\{[\mathbf{p}^i_{\text{wp}}, \boldsymbol{\beta}^i_{\text{wp}}]\}_{i=1}^m$	A motion profile with m waypoints
$\boldsymbol{\eta}$	Coordinate for applying the actions
$\mathbf{InvKin}(\cdot)$	Inverse Kinematic
$J^{-1}(\cdot)$	Jacobian of the inverse kinematic function
$(p^i_{\max})^k$	Maximum local position error around the i^{th} waypoint at the k^{th} iteration
$(\beta^i_{\max})^k$	Maximum orientation position error around the i^{th} waypoint at the k^{th} iteration

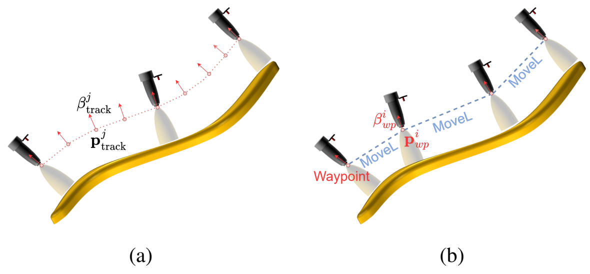

Fig. 2: Example of a trajectory-tracking problem: (a): Trajectory to track. (b) Fit the trajectory using MoveL primitives.

B. Motion Primitives and Their Execution

For most industrial robots, motion profiles can be programmed using several types of motion primitives, most commonly:

- MoveL: Both the position $\mathbf{p}$ and orientation β are linear functions of time.
- MoveC: The position $\mathbf{p}$ follows a circular arc while the orientation β is a linear functions of time.
- MoveJ: The joint angles $\mathbf{q}$ are linear functions of time.

For simplicity, we only train on MoveL primitives and evaluate on MoveL and MoveJ. A MoveL primitive is parameterized by its starting and target positions and orientation. A MoveC primitive would need an additional parameter – the circle's radius. A MoveJ primitive is specified in joint angle space instead of the Cartesian space on which the target position $\mathbf{p}$ and orientation β are specified.

At execution, the robot controller blends the motion primitives into a smooth trajectory [10] through a process that is usually vendor-specific and not accessible to the users. *Blending zone* is a hyperparameter that indicates the size of the blended region around each waypoint. Larger blending zones increase execution speed but result in worse tracking accuracy. Conversely, smaller blending zones improve tracking accuracy but may require lower execution speed because of sharp turns. This paper considers a fixed blending zone that spans a 10mm region around each waypoint.

C. Specification

We impose the following specifications on the trajectory-tracking task:

1) **Positional tracking accuracy**: the maximum positional tracking error of TCP is less than 0.5 mm:

$$e^{p}_{\text{max}} = \max_{j} \left\{ \|\mathbf{p}^{j}_{\text{execute}} - \mathbf{p}^{j}_{\text{track}}\|_2 \right\}_{j=1}^{n} \leq 0.5\text{mm}. \quad (1)$$

2) **Orientation tracking accuracy**: the maximum orientation tracking error of tool z-axis is less than 3°:

$$e^{\beta}_{\text{max}} = \max_{j} \left\{ \|\boldsymbol{\beta}^{j}_{\text{execute}} - \boldsymbol{\beta}^{j}_{\text{track}}\|_2 \right\}_{j=1}^{n} \leq 3^\circ. \quad (2)$$

3) **Speed uniformity**: the standard deviation of the TCP speed is less than 5% of the average TCP speed:

$$\begin{aligned} \{\lambda^j\}_{j=1}^{n} &= \left\{ \frac{\|\mathbf{p}^{j}_{\text{execute}} - \mathbf{p}^{j-1}_{\text{execute}}\|_2}{\Delta t^j} \right\}_{j=1}^{n}, \\ \text{var}(\{\lambda^j\}_{j=1}^{n}) &\leq 0.05 \cdot \text{mean}(\{\lambda^j\}_{j=1}^{n}). \end{aligned} \quad (3)$$

where Δt^j is the time interval between the jth and $(j+1)$th points.

Algorithm 1 Motion Profile Optimization

```
Input: fully resolved trajectory $\mathbf{T}_{\text{track}}$
$v^0 \leftarrow$ Initial speed command
$v_{\text{min}}, v_{\text{max}} \leftarrow$ Bounds of bi-section search
$c \leftarrow 0$
while maximum outer iterations not reached do
    ($\downarrow$ Inner: Minimize execution error using IWA)
    (Motion profile)$^c$, execution results $\leftarrow$ IWA($v^c$, $\mathbf{T}_{\text{track}}$)
    ($\downarrow$ Outer: Maximize average execution speed using bi-section search)
    if execution results satisfy all specifications then
        $v^{c+1} \leftarrow \frac{1}{2}(v_{\text{max}} + v^c)$
        $v_{\text{min}} \leftarrow v^c$
    else
        $v^{c+1} \leftarrow \frac{1}{2}(v_{\text{min}} + v^c)$
        $v_{\text{max}} \leftarrow v^c$
    end if
    $c \leftarrow c + 1$
end while
return Highest $v^c$ and (Motion profile)$^c$
```

III. Motion Profile Optimization

A. Formulation

Conceptually, we can write the Robot Execution block in Fig. 1 as a function

$$\texttt{Execute}(\{\mathbf{p}^i_{\text{wp}}, \boldsymbol{\beta}^i_{\text{wp}}\}_{i=1}^{m}, v) \mapsto \mathbf{T}_{\text{execute}},$$

where $\{[\mathbf{p}^i_{\text{wp}}, \boldsymbol{\beta}^i_{\text{wp}}]\}_{i=1}^{m}$ are the m waypoints in the motion profile and v is the commanded speed. The `Motion Profile Optimization` step solves the optimization problem

$$\begin{aligned} &\max_{\{\mathbf{p}^i_{\text{wp}}, \boldsymbol{\beta}^i_{\text{wp}}\}_{i=1}^{m}, v} \text{mean}(\{\lambda^j\}_{j=1}^{n}) \\ &\textbf{s.t.} \max_{j} \left\{ \|\mathbf{p}^{j}_{\text{execute}} - \mathbf{p}^{j}_{\text{track}}\|_2 \right\}_{j=1}^{n} \leq 0.5\text{mm} \\ &\quad \max_{j} \left\{ \|\boldsymbol{\beta}^{j}_{\text{execute}} - \boldsymbol{\beta}^{j}_{\text{track}}\|_2 \right\}_{j=1}^{n} \leq 3^\circ \\ &\quad \text{var}(\{\lambda^j\}_{j=1}^{n}) \leq 0.05 \cdot \text{mean}(\{\lambda^j\}_{j=1}^{n}) \\ &\textbf{where } \mathbf{T}_{\text{execute}} = \texttt{Execute}(\{\mathbf{p}^i_{\text{wp}}, \boldsymbol{\beta}^i_{\text{wp}}\}_{i=1}^{m}, v) \\ &\quad \{\mathbf{p}^{j}_{\text{execute}}, \boldsymbol{\beta}^{j}_{\text{execute}}\}_{j=1}^{n} = \mathbf{T}_{\text{execute}} \\ &\quad \{\mathbf{p}^{j}_{\text{track}}, \boldsymbol{\beta}^{j}_{\text{track}}\}_{j=1}^{n} = \mathbf{T}_{\text{track}} \\ &\quad \{\lambda^j\}_{j=1}^{n} = \left\{ \frac{\|\mathbf{p}^{j}_{\text{execute}} - \mathbf{p}^{j-1}_{\text{execute}}\|_2}{\Delta t^j} \right\}_{j=1}^{n}. \end{aligned} \quad (4)$$

The `Motion Profile Optimization` step is a bi-level optimization process that maximizes the average TCP speed under the accuracy and uniformity constraints.The procedures are presented in Alg. 1. The inner problem optimizes $\{\mathbf{p}^i_{\text{wp}}, \boldsymbol{\beta}^i_{\text{wp}}\}_{i=1}^{m}$ by iteratively updating waypoint $\mathbf{p}^i_{\text{wp}}$ and $\boldsymbol{\beta}^i_{\text{wp}}$ for $i = 1, \ldots, m$

$$\begin{cases} (\mathbf{p}^i_{\text{wp}})^{k+1} = (\mathbf{p}^i_{\text{wp}})^{k} + (\Delta\mathbf{p}^i_{\text{wp}})^{k} \\ (\boldsymbol{\beta}^i_{\text{wp}})^{k+1} = (\boldsymbol{\beta}^i_{\text{wp}})^{k} + (\Delta\boldsymbol{\beta}^i_{\text{wp}})^{k} \end{cases}, \quad (5)$$

which minimizes the tracking errors while ignoring the speed uniformity constraint. We call this step *Iterative Waypoint Adjustment* (IWA). The outer problem maximizes the speed command v, using a bi-section search approach to satisfy all three constraints.

B. Gradient-Based Method

The *Multi-peak Gradient Descent* (MPGD) method [14] is a model-based non-learning approach that uses an approximated model to calculate the numerical gradient of waypoint adjustment and optimize waypoints based on gradient descent. To simplify computation, gradient calculations and waypoint adjustments perform only on the few waypoints around each peak tracking error. For the ith peak waypoint with position $\mathbf{p}^i_{\text{wp}}$ and orientation $\boldsymbol{\beta}^i_{\text{wp}}$, the gradient of adjusting the waypoint on the x axis with respect to the positional error is

$$\nabla e^i_x = \frac{d\|\mathbf{e}^i\|_2}{du^i_x} \approx \frac{\|\mathcal{M}(\mathbf{p}^i_{\text{wp}} + [\delta, 0, 0]^\top) - \mathbf{p}^{\psi_i}_{\text{track}}\|_2 - \|\mathbf{e}^i\|_2}{\delta}, \quad (6)$$

where $\mathcal{M}$ is a cubic spline interpolation model to predict the local execution trajectory given several waypoints around the

ith waypoint, δ is a small perturbation, and $\mathbf{e}^i = \mathbf{p}_{\text{track}}^{\psi^i} - \mathbf{p}_{\text{execute}}^{\psi^i}$. The same calculation applies to other axes and orientations. Then, $\left[(\Delta\mathbf{p}_{\text{wp}}^i)^k, (\Delta\boldsymbol{\beta}_{\text{wp}}^i)^k\right]$ is calculated using gradient descent and the waypoints are updated iteratively.

In experiments, the MPGD method may require many iterations to meet the tracking error specifications. At each iteration, the robot needs to execute the motion profile, where running robots can be time-consuming and expensive. For example, in simulation, it takes about 12 seconds to execute a trajectory. On a real robot, since the execution result may not be uniform due to physical uncertainty, the execution trajectory used for gradient computation is the average of five runs for a given motion profile.

IV. Reinforcement Learning Approach

A. *Overview of Data-Driven Approaches*

Motivated by the inefficiency of the current approaches, we use data-driven methods to reduce the number of iterations required to achieve specified execution errors and, therefore, reduce the cost of running this optimization process on robots. The IWA process can be modeled as a sequential decision-making problem to find a sequence of control inputs $\mathcal{U} = \left\{\left\{(\mathbf{u}^i)^k\right\}_{i=1}^{m}\right\}_{k=1}^{k_{\max}}$, where $(\mathbf{u}^i)^k = \left[(\Delta\mathbf{p}_{\text{wp}}^i)^k, (\Delta\boldsymbol{\beta}_{\text{wp}}^i)^k\right]$. In this paper, we define features, actions, and rewards for the problem and propose a method to find $\mathcal{U}$ using reinforcement learning.

An important finding from the MPGD method is that the effect of adjusting a waypoint on the execution errors is mostly local. Therefore, in our formulation, the iterative optimization method is formulated locally:

- Features are defined locally, around each waypoint.
- The policy applies to all waypoints, but generates actions for each waypoint separately. Each iteration adjusts all waypoints at the same time.
- Reward is calculated based on local maximum execution error.

Data-driven methods take local features as input and output $\mathbf{a}^i$ for the corresponding waypoint. Feature and reward calculations of IWA are presented in Alg. 2.

B. *Reinforcement Learning Formulation*

Features, actions, rewards, and termination conditions are formulated as the following:

- **State**: The trajectory to track and the current robot execution trajectory. The state is usually not directly accessible. We decompose the state into local state features.
- **State features**: Features are defined locally around each waypoint, i.e., between the previous and next waypoints. For the first waypoint, the local interval is between the first and second waypoint. For the last waypoint, the local interval is between the second-last and last waypoint. The number of points in the execution trajectory may vary depending on the data streaming frequency and length of the trajectory. Therefore, we first normalize the length of the local trajectory to 50 points using linear interpolation.

  The state features vector s^i for the ith waypoint has 313 elements from:

  ► $\bar{\boldsymbol{P}}_{\text{error}}^i \in \mathbb{R}^{50\times3}$: Local position error trajectory normalized by error value at the ith waypoint:

$$\bar{\boldsymbol{P}}_{\text{error}}^i = \left\{\mathbf{p}_{\text{error}}^{i,j}/\|\mathbf{e}^i\|_2\right\}_{j=1}^{50}. \tag{7}$$

  ► $\bar{\boldsymbol{P}}_{\text{target}}^i \in \mathbb{R}^{50\times3}$: Local target position trajectory normalized using *Principle Component Analysis* (PCA). PCA removes the effect of trajectory orientation and only maintains shape information:

$$\bar{\boldsymbol{P}}_{\text{target}}^i = \text{PCA}\left(\left\{\mathbf{p}_{\text{target}}^{i,j}\right\}_{j=1}^{50}\right) \Big/ \|\mathbf{p}_{\text{target}}^{i,50} - \mathbf{p}_{\text{target}}^{i,1}\|_2. \tag{8}$$

  ► $\mathbf{q}^i \in \mathbb{R}^{1\times6}$: Robot pose at the ith waypoint:

$$\mathbf{q}^i = \textbf{InvKin}\left(\begin{bmatrix} \mathbf{p}_{\text{wp}}^i \\ \boldsymbol{\beta}_{\text{wp}}^i \end{bmatrix}\right). \tag{9}$$

  ► $\mathbf{e}^i \in \mathbb{R}^{1\times3}$: Execution error at the ith waypoint:

$$\mathbf{e}^i = \mathbf{p}_{\text{track}}^{\psi^i} - \mathbf{p}_{\text{execute}}^{\psi^i}. \tag{10}$$

  ► $F_{\text{Jac}} \in \mathbb{R}^1$: Effect of adjusting the waypoint on robot joints. A larger value implies more joint movements:

$$F_{\text{Jac}}^i \leftarrow \|J^{-1}(\mathbf{q}^i)[\mathbf{e}^i, \mathbf{0}]^\top\|_2. \tag{11}$$

  ► $F_{\text{position}}^i \in \mathbb{R}^1$: position of the ith waypoint in the local interval:

$$F_{\text{position}} \leftarrow (\psi^i - \psi^{i-1})/(\psi^{i+1} - \psi^{i-1}). \tag{12}$$

  ► $F_{\text{flag}}^i \in \mathbb{R}^2$: Boolean variables that indicate the first and the last waypoint:

$$F_{\text{flag}}^i = \begin{cases} [1,0] & \text{if } i = 1 \\ [0,1] & \text{if } i = m \\ [0,0] & \text{otherwise} \end{cases}. \tag{13}$$

- **Action**: $\mathbf{a}^i \in \mathbb{R}^{1\times3}$ represent positional adjustments for a waypoint in Cartesian space. Note that the orientations of waypoints are fixed after initialization. We address the orientation error in the reward function. Instead of using Cartesian coordinates, we define the relative coordinates for applying actions:
  - **Axis 1**: Towards the direction of the error vector at the waypoint. Its unit vector is $\eta_1 = \mathbf{e}^i/\|\mathbf{e}^i\|_2$.
  - **Axis 2**: Towards the previous waypoint. Its unit vector is $\eta_2 = (\mathbf{p}_{\text{wp}}^{i-1} - \mathbf{p}_{\text{wp}}^i)/\|\mathbf{p}_{\text{wp}}^{i-1} - \mathbf{p}_{\text{wp}}^i\|_2$.
  - **Axis 3**: Cross product of unit vectors of axis 1 and axis 2. Its unit vector is $\eta_3 = \eta_1 \times \eta_2$.

  The control inputs are

$$\Delta\mathbf{p}_{\text{wp}}^i = \mathbf{a}^i \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \end{bmatrix} \cdot \|\mathbf{e}^i\|_2, \ \Delta\boldsymbol{\beta}_{\text{wp}}^i = \mathbf{0}, \tag{14}$$

  where $\eta_j \in \mathbb{R}^{1\times3}$ for $j = 1, 2, 3$.
- **Reward**: Rewards show improvement in maximum local execution error compared to the previous iteration, normalized by the maximum execution error at iteration 0.

Let $(p^i_{\max})^k$ and $(\beta^i_{\max})^k$ denote the maximum positional and orientation error in the local region of i^{th} waypoint at the k^{th} iteration. The reward consists of the following components:

▶ Position error (r^i_p): Relative improvement in position error:

$$r^i_p \leftarrow \mathbf{clip}\big(((p^i_{\max})^k - (p^i_{\max})^{k-1})/(p^i_{\max})^0, -10, 10\big). \tag{15}$$

▶ Orientation error (r^i_β): Relative improvement in orientation error:

$$r^i_\beta \leftarrow \mathbf{clip}\big(((\beta^i_{\max})^k - (\beta^i_{\max})^{k-1})/(\beta^i_{\max})^0, -10, 10\big). \tag{16}$$

▶ Failure (r^i_{fail}): If the maximum local position error exceeds 5mm:

$$r^i_{\text{fail}} = -100 \textbf{ if } (p^i_{\max})^k > 5\text{mm} \textbf{ else } 0. \tag{17}$$

▶ Success (r^i_{success}): If the maximum local position error is below 0.25mm:

$$r^i_{\text{success}} = 10 \textbf{ if } (p^i_{\max})^k < 0.25\text{mm} \textbf{ else } 0. \tag{18}$$

- **Termination**: In regular RL problems, failure and success also represent the termination of an episode. In our formulation, since a state is divided into multiple local segments, the local termination flag d indicates a local success or failure. However, instead of executing each local segment separately, the robot executes the entire trajectory. Therefore, the termination of an episode Φ is in a global view, based on the following criteria:
  - **Fail**: $\max_j \|\mathbf{p}^j_{\text{execute}} - \mathbf{p}^j_{\text{track}}\|_2 > 5$mm or robot joints are close to singularities (from simulated robot model or the robot controller feedback).
  - **Success**: $\min_j \|\mathbf{p}^j_{\text{execute}} - \mathbf{p}^j_{\text{track}}\| < 0.25$mm.
  - **Reaches iteration limit**: $k \geq k_{\max}$.

Deep learning methods are increasingly applied to robot control tasks [6] [18]. In this paper, we use a *Deep Reinforcement Learning* approach and train a policy to find $\mathbf{a}^i$ for each waypoint.

C. Reinforcement Learning Algorithm

At each iteration, $\mathbf{a}^i \in \mathbb{R}^3$ is a continuous action. Therefore, we use *Twin-Delayer Deep Deterministic Policy Gradient* (TD3) [19], an RL algorithm that has state-of-the-art performance on the continuous-action-space benchmarks.

Alg. 3 shows the procedure for training the RL policy, where Q_θ is the critic network with parameter θ, π_ϕ is the actor network with parameter ϕ, ρ denotes the update rate of target networks, and γ denotes the decay factor in Q learning. The policy $\pi_\phi(S) : s^i \mapsto \mathbf{a}^i$ represents RL action selection. We add Gaussian exploration noise to actions $\epsilon \sim \mathcal{N}$ as the exploration strategy during training. Note that this noise is removed during evaluation.

Algorithm 2 Iterative Waypoint Adjustment (IWA)

```
Input: speed command $v$, $\mathbf{T}_{\text{track}}$
$\{s^i\}_{i=1}^m \leftarrow$ INITIALIZATION($\mathbf{T}_{\text{track}}$)
while $k < k_{\max}$ and not $\Phi$ do
    $\{\mathbf{a}^i\}_{i=1}^m \leftarrow$ Policy
    $\{s^i_n, r^i, d^i\}_{j=1}^m, \Phi \leftarrow$ IWA-ITERATION($\{\mathbf{a}^i\}_{i=1}^m$)
    $(e^p_{\max})^k, (e^\beta_{\max})^k, \{(v^j)^k\}_{j=1}^n \leftarrow$ Spec($\mathbf{T}_{\text{execute}}, \mathbf{T}_{\text{track}}$)
    $\{s^i\}_{i=1}^m \leftarrow \{s^i_n\}_{i=1}^m$
end while
return $\{[(\mathbf{p}^i_{\text{wp}})^k, (\boldsymbol{\beta}^i_{\text{wp}})^k]\}_{i=1}^m$ with the lowest positional error and the corresponding $(e^p_{\max})^k, (e^\beta_{\max})^k, \{(\lambda^j)^k\}_{j=1}^n$.
INITIALIZATION (input: $\mathbf{T}_{\text{track}}$)
    $k = 0$
    $\{[(\mathbf{p}^i_{\text{wp}})^k, (\boldsymbol{\beta}^i_{\text{wp}})^k]\}_{i=1}^m \leftarrow$ Sampled from $\mathbf{T}_{\text{track}}$
    Set primitive type to MoveL
    $\{s^i\}_{i=1}^m \leftarrow$ GET-FEATURE
    for $i = 1, \ldots, m$ do
        $\{\mathbf{p}^{i,j}_{\text{error}}, \boldsymbol{\beta}^{i,j}_{\text{error}}\}_{j=1}^{50} \leftarrow \mathbf{T}^i_{\text{error}}$
        $(p^i_{\max})^0 \leftarrow \max_j(\|\mathbf{p}^{i,j}_{\text{error}}\|_2)$
        $(\beta^i_{\max})^0 \leftarrow \max_j(\|\boldsymbol{\beta}^{i,j}_{\text{error}}\|_2)$
    end for
    return $\{s^i\}_{i=1}^m$
ITERATION (input: $\mathcal{A} = [\mathbf{a}^1, \ldots, \mathbf{a}^m]$)
    $(\mathbf{p}^i_{\text{wp}})^{k+1} \leftarrow$ Apply $\mathbf{a}^i$ to $(\mathbf{p}^i_{\text{wp}})^k$ for $i = 1, \ldots, m$
    $k \leftarrow k + 1$
    $\{s^i\}_{i=1}^m \leftarrow$ GET-FEATURE
    $\{r^i\}_{i=1}^m, \{d^i\}_{i=1}^m, \Phi \leftarrow$ CALCULATE-REWARD
    return $\{s^i\}_{i=1}^m, \{r^i\}_{i=1}^m, \{d^i\}_{i=1}^m, \Phi$
GET-FEATURE
    $\mathbf{T}_{\text{execute}} \leftarrow$ Execute $(\{[(\mathbf{p}^i_{\text{wp}})^k, (\boldsymbol{\beta}^i_{\text{wp}})^0]\}_{i=1}^m, v)$
    Match index of $\mathbf{T}_{\text{execute}}$ and $\mathbf{T}_{\text{track}}$
    $\{[\mathbf{p}^j_{\text{execute}}, \boldsymbol{\beta}^j_{\text{execute}}]\}_{j=1}^m \leftarrow \mathbf{T}_{\text{execute}}$
    $\psi^i \leftarrow \text{argmin}_j \|\mathbf{p}^j_{\text{execute}} - (\mathbf{p}^i_{\text{wp}})^k\|_2$ for $i = 1, \ldots, 100$
    $\psi^0, \psi^{101} \leftarrow 0, \text{EOT}$
    for $i = 1, \ldots, m$ do
        $\mathbf{T}^i_{\text{target}} \leftarrow \mathbf{T}_{\text{track}}[\psi^{i-1} : \psi^{i+1}]$
        $\mathbf{T}^i_{\text{error}} \leftarrow \mathbf{T}^i_{\text{target}} - \mathbf{T}_{\text{execute}}[\psi^{i-1} : \psi^{i+1}]$
        $\{\mathbf{p}^{i,j}_{\text{error}}, \boldsymbol{\beta}^{i,j}_{\text{error}}\}_{j=1}^{50} \leftarrow \mathbf{T}^i_{\text{error}}$
        $\{\mathbf{p}^{i,j}_{\text{target}}, \boldsymbol{\beta}^{i,j}_{\text{target}}\}_{j=1}^{50} \leftarrow \mathbf{T}^i_{\text{target}}$
        $\bar{\boldsymbol{P}}^i_{\text{target}}, \bar{\boldsymbol{P}}^i_{\text{error}}, \mathbf{q}^i, \mathbf{e}^i, F^i_{\text{Jac}}, F^i_{\text{position}}, F^i_{\text{flag}} \leftarrow$ Eq. 7 to 13
        $s^i \leftarrow (\bar{\boldsymbol{P}}^i_{\text{target}}, \bar{\boldsymbol{P}}^i_{\text{error}}, \mathbf{q}^i, \mathbf{e}^i, F_{\text{Jac}}, F_{\text{position}}, F_{\text{flag}})$
    end for
    return $\{s^i\}_{i=1}^m$
CALCULATE-REWARD
    for $i = 1, \ldots, m$ do
        $\{\mathbf{p}^{i,j}_{\text{error}}, \boldsymbol{\beta}^{i,j}_{\text{error}}\}_{j=1}^{50} \leftarrow \mathbf{T}^i_{\text{error}}$
        $(p^i_{\max})^k \leftarrow \max_j(\|\mathbf{p}^{i,j}_{\text{error}}\|_2)$
        $(\beta^i_{\max})^k \leftarrow \max_j(\|\boldsymbol{\beta}^{i,j}_{\text{error}}\|_2)$
        $r^i_p, r^i_\omega, r^i_{\text{fail}}, r^i_{\text{success}} \leftarrow$ Eq. 15 to Eq. 18
        $r^i = -r^i_p - 0.5 r^i_\omega + r^i_{\text{fail}} + 0.8^k \cdot r^i_{\text{success}}$
        $d^i = (p^i_{\max})^k > 5$ or $(p^i_{\max})^k < 0.25$ or $k \geq 10$
    end for
    $\Phi \leftarrow$ Termination conditions
    return $\{r^i\}_{i=1}^m, \{d^i\}_{i=1}^m, \Phi$
```

Algorithm 3 RL Training

```
Randomly initialize actor and critic networks $\pi_\phi, Q_{\theta_1}, Q_{\theta_2}$
Copy parameters and create target networks: $\pi_{\phi'}, Q_{\theta'_1}, Q_{\theta'_2}$
Initialize empty replayer buffer $\mathcal{D}$
$\mathbf{T}_{\text{track}} \leftarrow$ Randomly select a trajectory from the dataset
$\{s^i\}_{i=1}^m \leftarrow$ IWA-INITIALIZATION($\mathbf{T}_{\text{track}}$)
$t \leftarrow 1$
while maximum training step not reached do
    $\{\mathbf{a}^i\}_{i=1}^m \leftarrow \mathbf{clip}(\pi_\phi(\{s^i\}_{i=1}^m) + \epsilon, -2, 2), \epsilon \sim \mathcal{N}$
    $\{s_n^i, r^i, d^i\}_{i=1}^m, \Phi \leftarrow$ IWA-ITERATION($\{\mathbf{a}^i\}_{i=1}^m$)
    Save $\{s^i, \mathbf{a}^i, r^i, d^i, s_n^i\}_{i=1}^m$ to $\mathcal{D}$
    Sample mini-batch of $N$ tuples $(s, \mathbf{a}, r, d, s_n)$ from $\mathcal{D}$
    $\mathbf{a}_n \leftarrow \mathbf{clip}\big(\pi_{\phi'}(s_n) + \mathbf{clip}(\epsilon, -0.5, 0.5), -2, 2\big), \epsilon \sim \mathcal{N}$
    $q'_1, q'_2 \leftarrow Q_{\theta'_1}(s_n, \mathbf{a}_n), Q_{\theta'_2}(s_n, \mathbf{a}_n)$
    $q_{\text{target}} \leftarrow r + (1-d)\gamma \min(q'_1, q'_2)$
    $q_1, q_2 \leftarrow Q_{\theta_1}(s, \mathbf{a}), Q_{\theta_2}(s, \mathbf{a})$
    Update critics by one step of gradient descent:
        $\nabla_{\theta_i} \frac{1}{N} \sum_{(s,a,r,d,s_n) \in B} (q_i - q_{\text{target}})^2$ for $i = 1, 2$
    (Update sample weights of $\mathcal{D}$ if $\mathcal{D}$ is PER)

    if $t$ mod policy_update_frequency = 0 then
        Update actor by one step of gradient descent:
            $-\nabla_\phi \frac{1}{|B|} \sum_{s \in B} Q_{\theta_1}(s, \pi_\phi(s))$
        Soft update target networks:
            $\phi' \leftarrow \rho\phi' + (1-\rho)\phi$
            $\theta'_i \leftarrow \rho\theta'_i + (1-\rho)\theta_i$ for $i = 1, 2$
    end if
    $\{s^i\}_{i=1}^m \leftarrow \{s_n^i\}_{i=1}^m$
    if $\Phi$ then
        $\mathbf{T}_{\text{track}} \leftarrow$ randomly select a trajectory from the dataset
        $\{s^i\}_{i=1}^m \leftarrow$ IWA-INITIALIZATOIN($\mathbf{T}_{\text{track}}$)
    end if
    $t \leftarrow t + 1$
end while
```

D. Addressing Outliers and Data Imbalance Issue

In experiments, we find outliers in the training data causing failures during training. For transition tuples in $\mathcal{D}$, most of the feature vectors s^i are tightly distributed with some outliers. In the actor-critic method, estimating the Q values of each data point is a regression task. The performance of policy largely depends on the accurate estimation of Q values.

Most machine learning and deep RL models use *Rectified Linear Units* (`ReLU`) neural networks since they are efficient in gradient computation and have been demonstrated to be effective on most problems. However, `ReLU` activation function is sensitive to outliers. Therefore, in our implementation, we use the *Hyperbolic Tangent* (`Tanh`) activation function which has zero gradient at large input values, which results in stable training.

Since `Tanh` activation functions saturates at large input values, the training result may be biased towards the main distribution. Our experiment results show that the learned policy can reduce error for most of the waypoints but fail for several waypoints with high local curvature. The training should use the outlying data points more frequently to balance the training data. We use the *Prioritized Experience Replay* (PER) [20] method to relax the issue of imbalanced data and overfitting.

V. Experiment

A. Test Trajectories

In this paper, we use two types of trajectories. Trajectory type 1 is a multi-frequency sine trajectory on a parabolic surface, representing a high curvature spatial trajectory (Fig. 3). Trajectory type 2 is the leading edge of a fan blade, which is used for cold spraying applications in the industry (Fig. 4), extracted from a CAD model.

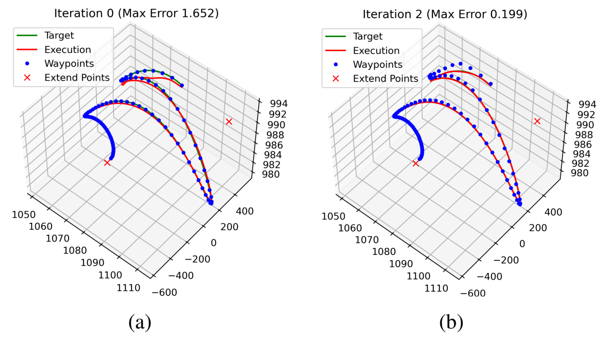

Fig. 3: Example of trajectory type 1 at pose 1 for: (a): baseline (100 MoveL). (b) optimized motion profile using learned RL policy.

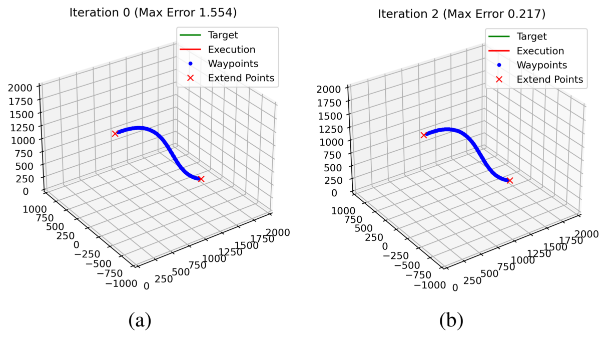

Fig. 4: Example of trajectory type 2 at pose 1 for (a): baseline (100 MoveL) (b) optimized motion profile using learned RL policy.

B. Setup and Baseline

In experiments, we use RobotStudio [15], an ABB robot dynamics simulator, to run an ABB 6640 robot [21] for training. The performance of the trained policy is evaluated on both the simulated robot and the real robot.

The training dataset contains 300 randomly generated type 1 trajectories in three poses. The evaluation dataset contains seven manually selected trajectories with different curvatures and poses, where higher curvature implies higher difficulty.

TABLE II: Iterative Waypoint Adjustment - RL Evaluation

		Speed Cmd.	Baseline	MPGD(Sim.)			RL(Sim.)			MPGD(Real)		RL (Real)	
Traj.	Pose	(Sim./Real)	Err.(Sim./Real)	Err.	Itr.	t_{opt}	Err.	Itr.	t_{opt}	Err.	Itr.	Err.	Itr.
Type 1 Hard	1	250/250	1.55/1.92	1.07	N/A	N/A	1.14	N/A	N/A	1.14	N/A	1.52	N/A
Type 1 Easy	1	250/250	0.80/1.02	0.20	3	95.45	0.23	1	9.61	0.20	7	0.23	1
Type 1 Medium	1	250/250	1.79/2.06	0.43	5	333.38	0.45	2	22.73	0.47	5	0.58	N/A
Type 1 (pose 1)	1	250/250	1.63/2.07	0.20	3	153.71	0.13	1	10.74	0.21	7	0.14	2
Type 1 (pose 2)	2	400/300	4.60/4.28	0.66	N/A	N/A	0.46	7	63.46	0.88	N/A	1.1	N/A
Type 1 (pose 3)	3	400/400	2.03/3.66	0.31	8	383.67	0.23	2	17.42	0.70	N/A	0.37	3
Type 2 (pose 1)	1	1100/750	1.74/1.31	0.21	14	642.66	0.22	2	17.68	0.26	6	0.24	1
Type 2 (pose 4)	4	1100/700	1.71/1.24	0.51	N/A	N/A	0.22	2	17.30	0.36	3	0.23	2
Type 2 (MoveJ)	1	1100/700	1.59/1.22	0.22	11	156.00	0.23	2	17.57	0.40	6	0.21	1

Trajectory type 1 at different poses has the standard curvature which is slightly lower than the medium curvature. Poses and their corresponding speed command are obtained from the redundancy resolution step in [14]. Note that trajectory type 2 is not included in the training dataset and we use them to demonstrate the learned policy can generalize to different curves, poses, and type of primitives.

All the trajectories are executed using the blending zone command $z10$. The initial motion profile contains $m = 100$ waypoints sampled uniformly from $\mathbf{T}_{\text{track}}$ connected by MoveL primitives, which is used as the baseline.

C. Performance

We train the RL policy for 5,000 iterations. The number of training episodes (number of trajectories) may vary depending on exploration.

For IWA, we compare the performance between the initial motion profile (baseline), MPGD, and the RL policy (RL) using three metrics:

- Err. (mm): Best maximum positional execution error obtained before reaching the maximum number of iterations.
- Itr.: Number of iterations it takes to achieve the 0.5mm specified execution error.
- t_{opt}(s): Total time to complete IWA optimization.

Note that $k_{\max} = 50$ for MPGD and $k_{\max} = 10$ for RL.

As evaluated in RobotStudio (Table II), the IWA using an RL policy outperforms MPGD, achieving 0.5 mm error specifications on more trajectories. The RL policy demonstrates a 4.2x improvement in `Itr.` and a 17.7x speedup in t_{opt}. It also reduces iterations to 1 or 2 for most trajectories. Type 2 trajectories confirm the policy's generalizability to various curve shapes, poses, and primitive types not present in the training dataset. Orientation errors are omitted as they consistently fall within the specified range.

On average, the MPGD method takes 60.86 seconds to select the actions at each iteration, and the RL policy takes 0.004 seconds. The total training time is 26.25 hours, with only 0.42 hours allocated for neural network training, suggesting that RobotStudio's efficiency can enhance training.

When running the real robot, the execution results may not be exactly repeatable since reproducibility is poor in industrial robots. In each iteration, a motion profile is executed five times, and $\mathbf{T}_{\text{execute}}$ is the average of five runs. Additionally, several trajectories need to be executed with lower speed commands to avoid blending errors reported by the robot controller. Fig. 5 shows the experiment setup on a real ABB 6640 robot. The green laser paths on the parts present the TCP trajectories.

With the trained RL policy, we run Alg. 1 for 10 iterations, using bi-section search to optimize the speed command. Table III summarizes the results of the complete `Motion Profile Optimization` process, obtained using RobotStudio. (*) indicates a violation of specifications. Our proposed method achieves all the specifications on all evaluation trajectories. Trajectories that satisfy both error and speed specification with only IWA result in higher average TCP speed. Other trajectories achieve uniform but lower speeds. Note that trajectory type 2 at pose 4 results in higher average speed. The optimized speed command is $v = 1089$ which reduces the effect of acceleration/deceleration and results in a higher average speed.

TABLE III: Speed Optimization with RL Policy

	IWA Only			IWA + Bi-section Search		
Traj.	Error mm	$\mu(\lambda)$ mm/s	$\sigma(\lambda)$ %	Error mm	$\mu(\lambda)$ mm/s	$\sigma(\lambda)$ %
Ty.1 Hard	1.14*	204.28	28.1*	0.49	91.13	3.0
Ty.1 Easy	0.23	251.96	0.3	0.31	408.78	4.8
Ty.1 Med.	0.45	224.74	20.0*	0.48	133.25	4.5
Ty.1 Po.1	0.48	248.91	3.9	0.11	258.89	4.9
Ty.1 Po.2	0.46	370.49	16.3*	0.48	223.37	0.9
Ty.1 Po.3	0.49	398.17	2.5	0.44	410.02	3.4
Ty.2 Po.1	0.22	1078.27	7.3*	0.19	991.25	4.8
Ty.2 Po.4	0.22	1080.86	7.8*	0.47	1085.25	3.4
Ty.2 MoveJ	0.23	1078.05	7.2*	0.49	998.18	4.9

D. Ablation Study of Iterative Waypoint Adjustment

In the ablation study, we remove each component in the proposed RL method to study their impact on performance. We include the following cases:

1) **RL−PER**: remove Prioritized Experience Replay.
2) **RL−$\boldsymbol{P}_{\text{target}}$**: remove local target position trajectory.
3) **RL−q − F_{Jac}**: remove features related to robot joints.
4) **RL−e**: remove information about error vector.

All cases are trained and evaluated using RobotStudio. Based on results in Table IV, removing one of these components reduces the generalizability of policies. Since deep

TABLE IV: IWA Ablation Study

	RL(Baseline)		RL−PER		RL−$\boldsymbol{P}_{\text{target}}$		RL−$\mathbf{q}$-F_{Jac}		RL−$\mathbf{e}$	
Traj.	Err.	Itr.	Err.	Itr.	Err.	It.r	Err.	Itr.	Err.	Itr.
Ty.1 Hard	1.14	N/A	1.09	N/A	1.18	N/A	1.30	N/A	1.33	N/A
Ty.1 Easy	0.23	1	0.17	1	0.23	1	0.25	1	0.24	1
Ty.1 Med.	0.45	2	0.43	2	0.45	2	0.42	2	0.47	3
Ty.1 Po.1	0.13	1	0.23	1	0.21	2	0.24	2	0.20	2
Ty.1 Po.2	0.46	7	0.96	N/A	1.39	N/A	2.00	N/A	1.56	N/A
Ty.1 Po.3	0.23	2	0.31	4	0.24	7	0.22	4	0.20	3
Ty.2 Po.1	0.24	2	0.24	2	0.18	2	0.17	2	0.23	3
Ty.2 Po.4	0.22	2	0.44	6	0.47	5	0.53	N/A	0.54	N/A
Ty.2 MoveJ	0.23	2	0.18	2	0.20	1	0.15	2	0.19	2

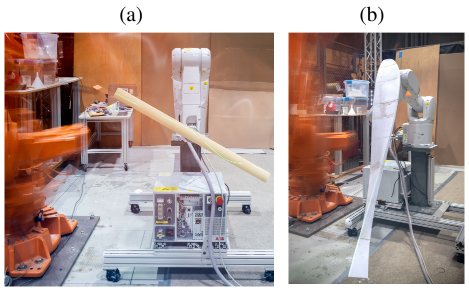

Fig. 5: Experiment setup of an ABB 6640 robot tracking curve (a) Type 1 (b) Type 2.

RL algorithms usually do not have convergence and reproducibility guarantee, the result can be impacted by training variation, which includes neural network initialization, exploration, learning rate, etc.

VI. Conclusion and Future Work

In this paper, we present a motion profile optimization method for enhancing accuracy and speed in trajectory-tracking tasks. We formulate the IWA process as a sequential decision-making problem and propose an RL-based policy training method. Our approach achieves the tracking specifications on a real industrial robot, outperforming existing methods in iterations needed to meet specifications and computation time per iteration, thereby saving time and robot operation costs.

In our experiments, we encounter imbalanced data under the current formulation. Although heuristic feature normalization and PER alleviate this issue, performance could be enhanced with appropriate normalization techniques. Additionally, while we only adjust waypoint positions, modifying waypoint orientation may further improve performance by offering more degrees of freedom, despite increased training difficulty.

Acknowledgment

The authors would like to thank Santiago Paternain, Eric Lu, Jonathan Fried, Pinghai Yang, and Jeff Schoonover for helpful discussions in our research. Research was sponsored by the ARM(Advanced Robotics for Manufacturing) Institute, through a grant from the Office of the Secretary of Defense and was accomplished under Agreement Number W911NF-17-3-0004. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of the Secretary of Defense or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] I. M. Nault, G. D. Ferguson, and A. T. Nardi, "Multi-axis tool path optimization and deposition modeling for cold spray additive manufacturing," *Additive Manufacturing*, vol. 38, p. 101779, 2021.

[2] X. Li, X. Li, S. S. Ge, M. O. Khyam, and C. Luo, "Automatic welding seam tracking and identification," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 9, pp. 7261–7271, 2017.

[3] K. Ma, X. Wang, and D. Shen, "Design and experiment of robotic belt grinding system with constant grinding force," in *2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, 2018, pp. 1–6.

[4] K. Ma, L. Han, X. Sun, C. Liang, S. Zhang, Y. Shi, and X. Wang, "A path planning method of robotic belt grinding for workpieces with complex surfaces," *IEEE/ASME Transactions on Mechatronics*, vol. 25, no. 2, pp. 728–738, 2020.

[5] Y. Xie, X. Tang, W. Meng, B. Ye, B. Song, J. Tao, and S. Q. Xie, "Iterative data-driven fractional model reference control of industrial robot for repetitive precise speed tracking," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 3, pp. 1041–1053, 2019.

[6] S. Yin, W. Ji, and L. Wang, "A machine learning based energy efficient trajectory planning approach for industrial robots," *Procedia CIRP*, vol. 81, pp. 429–434, 2019, 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019.

[7] C. Chang, K. Haninger, Y. Shi, C. Yuan, Z. Chen, and J. Zhang, "Impedance adaptation by reinforcement learning with contact dynamic movement primitives," in *2022 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2022, pp. 1185–1191.

[8] Q. Zhang, S.-R. Li, and X.-S. Gao, "Practical smooth minimum time trajectory planning for path following robotic manipulators," in *2013 American Control Conference*, 2013, pp. 2778–2783.

[9] T. Lipp and S. Boyd, "Minimum-time speed optimisation over a fixed path," *International Journal of Control*, vol. 87, no. 6, pp. 1297–1311, 2014.

[10] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity." in *Robotics: Science and Systems*, vol. 8, 2013, pp. 209–216 – 216.

[11] RoboDK Inc. , *Simulate Robot Applications: Program any Industrial Robot with One Simulation Environment*. [Online]. Available: https://www.robodk.com

[12] Hypertherm, Inc., *RobotMaster:CAD/CAM for robots (Off-Line Programming)*. [Online]. Available: https://www.robotmaster.com,

[13] OCTOPUZ, *RobotMaster:CAD/CAM for robots (Off-Line Programming)*. [Online]. Available: https://www.octopuz.com/,

[14] H. He, C.-l. Lu, Y. Wen, G. Saunders, P. Yang, J. Schoonover, J. Wason, A. Julius, and J. T. Wen, "High-speed high-accuracy spatial curve tracking using motion primitives in industrial robots," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 12 289–12 295.

[15] ABB Robotics, *Operating Manual RobotStudio*.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[17] J. Craig, *Introduction to Robotics: Mechanics and Control*, ser. Addison-Wesley series in electrical and computer engineering: control engineering. Pearson/Prentice Hall, 2005.

[18] H.-T. Nguyen and C. C. Cheah, "Analytic deep neural network-based robot control," *IEEE/ASME Transactions on Mechatronics*, vol. 27, no. 4, pp. 2176–2184, 2022.

[19] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, 2018, pp. 1582–1591.

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representation*, 2016.

[21] ABB Robotics, *Product specification IRB 6640*.