2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2025)
October 19-25, 2025. Hangzhou, China

2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 979-8-3315-4393-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/IROS60139.2025.11247526

Deep Reinforcement Learning-Based Trajectory Tracking Framework for 4WS Robots Considering Switch of Steering Modes

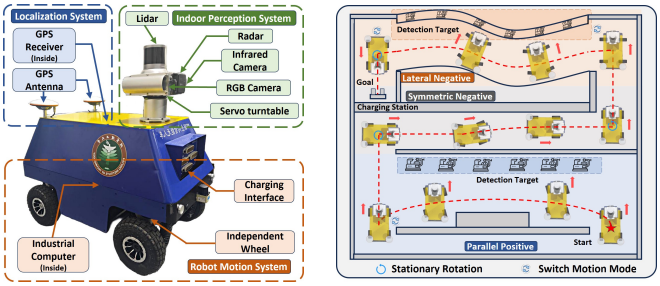
Runjiao Bao[1], Yongkang Xu[1], Lin Zhang[1], Haoyu Yuan[1], Jinge Si[1], Shoukun Wang[1], Tianwei Niu[1]*

***Abstract*— The application scenarios of automated robots are undergoing a paradigm shift from structured environments to unstructured, complex settings. In highly constrained settings like factory inspections or disaster rescue, conventional steering systems show clear drawbacks. While the four-wheel independent drive and independent steering (4WS) robot provides a variety of steering modes, which can effectively meet the needs of complex environments. However, how a 4WS robot autonomously selects different steering modes based on trajectory point information during trajectory tracking remains a challenging problem. This paper proposes a multi-modal trajectory tracking method considering the switch of steering modes, which decomposes the trajectory tracking task into two parts: mode decision-making and tracking control. The corresponding method is designed based on deep reinforcement learning. Additionally, a target trajectory random generator and corresponding training interaction environment are designed to train the model in a data-driven manner. In the designed scenario, our tracker achieve more than a 30% improvement in average tracking error across all motion modes compared with model predictive control, and the decider's average decision position error is less than 2 cm. Extensive experiments demonstrate that our method achieves superior tracking performance and real-time capabilities compared to current methods.**

I. Introduction

With the advancement of automation technology, the application scenarios of automated robots are undergoing a paradigm shift from structured environments to unstructured, complex settings [1], [2]. In highly constrained settings like factory inspections or disaster rescue, conventional steering systems show clear drawbacks: they have limited turning ability and insufficient degrees of freedom in trajectory planning [3]. Although mobile robots based on Mecanum wheels or omnidirectional wheels offer high maneuverability, they cannot be heavily loaded and their mechanism is vulnerable to damage [4], [5]. While the four-wheel independent drive and independent steering (4WS) robot provides a variety of steering modes, which can effectively meet the needs of complex environments. Each steering model imposes different constraints on the wheels to generate distinct motion characteristics [6].

Many scholars have studied trajectory tracking control strategies for 4WS robots. Setiawan et al. [7] proposed a symmetric negative steering trajectory tracking control strategy for 4WS robots based on the Backstepping method. Tan et al. [8] introduce a sliding mode controller for trajectory tracking tasks under negative steering strategies. Arvind [9] further investigate the dynamics of the steering system to reduce the turning radius of 4WS robots. Lin et al. [10] combine reinforcement learning with model predictive control (MPC) to build a trajectory tracking controller for negative steering. The study in [11] demonstrates that using an MPC controller enables the 4WS robot's negative steering mode to maintain excellent stability during high-speed trajectory tracking.



(a) (b)

Fig. 1. (a) Prototype of a 4WS robot, (b) Typical work scenarios of 4WS robot.

The above research mainly focuses on controlling single steering modes, particularly negative steering and its variants, without considering the switching of steering modes based on practical needs. Steering mode switching can be categorized into two types: stationary and non-stationary switching. The former will stop motion first and then switch modes, while the latter will switch modes dynamically during motion. Nguyen et al. [12] address the non-stationary switching problem of multiple steering modes in trajectory tracking by constructing a hybrid integer MPC controller. They used the big-M method and the linearized model to reformulate the non-linear mixed integer formula into a linear form, then using solvers to compute optimal steering modes and control parameters. Xu et al. [13] propose a multi-objective genetic algorithm (MOGA) based on Non-dominated Sorting Genetic Algorithm II (NSGA-II) for optimizing wheel trajectories in non-stationary steering mode switching. These studies focusing on non-stationary switching only consider switching between steering modes with similar motion constraints and do not resolve the uncontrollable motion issues caused by mode switching. When the target mode has a similar motion constraint to the current mode, the non-stationary switching may not have a large motion error, but for the mode with a large difference, this error may be large enough to affect the

[1] School of Automation, Beijing Institute of Technology, Beijing, China.
* Corresponding authors: Tianwei Niu (ntwbit@bit.edu.cn).

This work was supported by the National Natural Science Foundation of China under Grant 62473044, and the BIT Research and Innovation Promoting Project under Grant 2024YCXY037.

safety of the robot. In this case, stationary switching will be more suitable. For example, Fig. 1 shows a 4WS inspection robot and its typical inspection scenario. In this scenario, the target trajectory includes orientation requirements for the robot, and a combination of multiple steering modes is necessary to complete the trajectory tracking successfully.

Therefore, how a 4WS robot autonomously selects different steering modes based on waypoint information during trajectory tracking remains a challenging problem. To address this, Deep Reinforcement Learning (DRL) is a promising candidate. DRL has been widely applied in robotic trajectory tracking. Wang et al. [14] integrated DRL with pure pursuit algorithm to construct a tracker. Zhao et al. [15] utilized lateral and longitudinal errors as state variables to develop an end-to-end trajectory tracking system. Owing to its representational learning ability and exploratory characteristics [16], DRL methods can autonomously make decisions based on the waypoint information of the reference trajectory. Although DRL has demonstrated robust performance and adaptability in single-modal tracking, existing studies have yet to explore the various steer modes of 4WS robots.

In this paper, we adopt a hierarchical DRL approach to execute both trajectory tracking tasks and mode switching decisions. We have constructed an iterative training environment that considers all actuator constraints and applies multiple randomizations to ensure the feasibility of deploying our method on a real 4WS platform. Due to the fast inference performance of the DRL process, the method we implemented exhibits excellent real-time performance. Furthermore, we conducted extensive experiments to validate the effectiveness of our approach. Specifically, our contributions include the design of a DRL-based steering mode selection strategy that autonomously selects the optimal steering configuration based on the target state, the development of a DRL-based trajectory tracking method compatible with various steering modes, and the construction of a training environment incorporating trajectory randomization and domain randomization techniques to enhance real-world deployment feasibility.

II. Problem Formulation

A. 4WS Robot Modelling

The 4WS system is illustrated in Fig. 2, where it allows each of the robot's four wheels to steer at different angles. The coordinates of each wheel in the robot's local coordinate system can be expressed as follows:

$$x_i=(-1)^{\lfloor (i-1)/2 \rfloor}L, \quad y_i=(-1)^{i-1}W, \quad (i=1,2,3,4), \tag{1}$$

Assuming the four wheels are at the same speed, kinematic analysis of the motion of the four wheels with respect to the robot as a whole can yield a kinematic model of the robot:

$$\begin{cases} \dot{x} = \frac{1}{4}\sum_{i=1}^{4} v\cos(\eta_i+\theta) \\ \dot{y} = \frac{1}{4}\sum_{i=1}^{4} v\sin(\eta_i+\theta) \\ \dot{\theta} = \omega = \sum_{i=1}^{4} \frac{v(-y_i\cos\eta_i + x_i\sin\eta_i)}{4{x_i}^2+4{y_i}^2}, \end{cases} \tag{2}$$

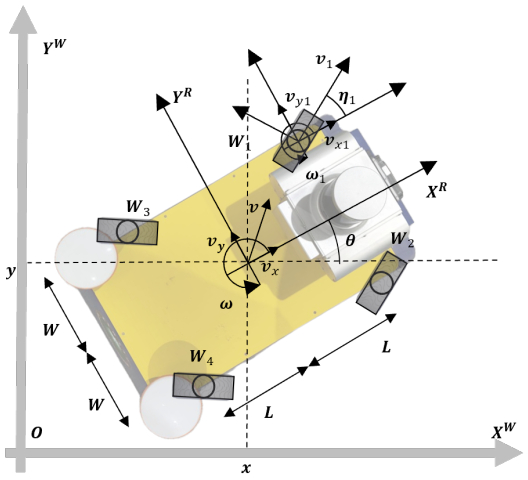


Fig. 2. 4WS robot kinematic model.

where state $\{x, y, \theta, v\}$ represents the robot's position, orientation and velocity, and input $\{a, \eta_1, \eta_2, \eta_3, \eta_4\}$ represents the wheel acceleration and the steering angles of each wheel.

This continuous model may suffer from distortion under discrete simulation conditions. Therefore, using a first-order Taylor expansion and Euler discretization method, we linearize and discretize the above model to obtain the system state equation:

$$X_{k+1} = AX_k + BU_k + C, \tag{3}$$

where the matrices A, B, and C are defined as:

$$A = \begin{bmatrix} 1 & 0 & -\alpha v^* \sum_{i=1}^{4} \sin\zeta_i^* & \alpha \sum_{i=1}^{4} \cos\zeta_i^* \\ 0 & 1 & \alpha v^* \sum_{i=1}^{4} \cos\zeta_i^* & \alpha \sum_{i=1}^{4} \sin\zeta_i^* \\ 0 & 0 & 1 & \alpha \sum_{i=1}^{4} \frac{-y_i\cos\eta_i^* + x_i\sin\eta_i^*}{4x_i^2+4y_i^2} \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$B = \begin{bmatrix} 0 & B_{12} & B_{13} & B_{14} & B_{15} \\ 0 & B_{22} & B_{23} & B_{24} & B_{25} \\ 0 & B_{32} & B_{33} & B_{34} & B_{35} \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$C = \begin{bmatrix} \alpha v^* \sum_{i=1}^{4} \zeta_i^* \sin(\zeta_i^*) \\ -\alpha v^* \sum_{i=1}^{4} \zeta_i^* \cos(\zeta_i^*) \\ -\alpha v^* \sum_{i=1}^{4} (\eta_i^* \frac{y_i\cos\eta_i^* + x_i\sin\eta_i^*}{4x_i^2+4y_i^2}) \\ 0 \end{bmatrix}. \tag{4}$$

Here, the elements of matrix B are defined as:

$$\begin{cases} B_{1i} = -\frac{1}{4}v^*\Delta t\sin(\zeta_i^*), \\ B_{2i} = \frac{1}{4}v^*\Delta t\cos(\zeta_i^*), \\ B_{3i} = \frac{1}{4}v^*\Delta t\frac{y_i\sin\eta_i^* + x_i\cos\eta_i^*}{4x_i^2+4y_i^2}, \end{cases} \tag{5}$$

where $\alpha = -\frac{1}{4}\Delta t, \zeta_i = \theta + \eta_i$. $X_k = [x_k, y_k, \theta_k, v_k]^T$ represents the robot's position, orientation, and velocity at time step k, and $U_k = [a_k, \eta_k]$ represents the robot's acceleration and the steering angles of the four wheels at time step k. The matrices A, B, and C are derived by linearizing the system at $[X^*, U^*]$.

In addition, to meet the robot's actuator performance, we considered the additional constraints defined by:

$$\begin{cases} X_{\min} \le X_k \le X_{\max} \\ U_{\min} \le U_k \le U_{\max} \\ |U_{k+1} - U_k| \le \Delta U_{\max}. \end{cases} \tag{6}$$

B. 4WS Steering Modes

The 4WS robot, due to the high degrees of freedom of its wheel mechanisms, has a variety of steering modes suitable for different scenarios, the typical of which is illustrated in Fig. 3. We focus on four typical steering modes: symmetrical negative steering, parallel positive steering, lateral negative steering, and stationary rotation. These four steering modes enable the robot to meeting the maneuverability requirements in various exceptional situations.

In the symmetric negative steering mode, the left and right wheel angles of the robot are the same, and the front and rear wheel angles are opposite. In parallel positive steering mode, the robot's four wheels turn at the same angle. Lateral negative steering can cause the robot to remain facing tangential to the direction of motion as it moves. Assuming a control input angle of η, the robot's motion constraints are:

$$\begin{cases} \eta_1 = \eta_3 = \eta, \\ \eta_2 = \eta_4 = \frac{\eta_1}{|\eta_1|}(\pi - |\eta_1|). \end{cases} \tag{7}$$

In stationary rotation mode, wheel angles only depend on the inherent properties of the robot. The robot's motion constraints are:

$$\begin{cases} \eta_1 = -\eta_3 = \pi - \arctan\left(\frac{L}{W}\right), \\ \eta_2 = -\eta_4 = \arctan\left(\frac{L}{W}\right). \end{cases} \tag{8}$$

C. Trajectory Tracking Optimization Formulation

To clearly describe the trajectory tracking problem, we define the optimization objective function as:

$$\begin{aligned} \min_{X_k, U_k} \quad & \sum_{i=0}^{N_p-1} \Big(E_{X,i}^\top Q E_{X,i} + E_{U,i}^\top R E_{U,i} \\ & + M_{U,i}^\top \Gamma M_{U,i} \Big) + E_{X,N_p}^\top P E_{X,N_p} \\ \text{subject to} \quad & E_{X,i} = \hat{X}_i - X_{k+i}^{\text{ref}}, \quad i = 0, \dots, N_p, \\ & E_{U,i} = \hat{U}_i - U_{k+i}^{\text{ref}}, \quad i = 0, \dots, N_p, \\ & M_{U,i} = \hat{U}_{i+1} - \hat{U}_i, \quad i = 0, \dots, N_p, \\ & \text{state constraints from Eq. (3)}, \\ & \text{Actuator constraint from Eq. (6)}, \end{aligned} \tag{9}$$

where, $Q \in \mathbb{R}^{4\times4}$, $R \in \mathbb{R}^{2\times2}$, $\Gamma \in \mathbb{R}^{2\times2}$, and $P \in \mathbb{R}^{4\times4}$ are the positive semi-definite weight matrices for the state, input, input change, and terminal state, respectively. Variables with a hat represent the predicted values at step i within the prediction horizon.

We need to optimize the solution of this objective function on the premise that the multiple constraints mentioned above are satisfied. Depending on the steering mode, we also need to decide which steering mode to adopt as the constraint of the optimization problem. The current standard method [12] is to use the big-M approach to construct the Mixed-Integer Quadratic Programming (MIQP) for a solution. However, this method of motion constraint switching takes effect instantaneously. In fact, the actual robot's steering mode switching takes time, which will cause the robot's motion to be uncontrolled during the switch. Switching between modes with significant constraint gaps can cause serious safety problems. This optimization problem will become more complicated with more optimization goals and constraints if it is desired that the robot be able to stand still before performing mode switching.

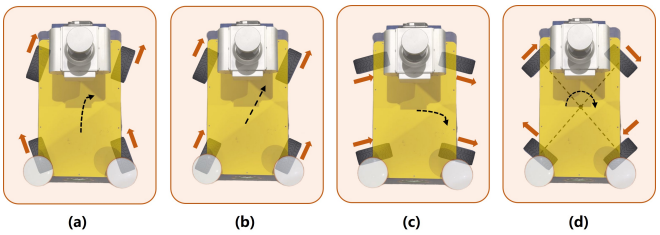


Fig. 3. 4WS robot steering modes: (a) Symmetrical negative steering. (b) Parallel positive steering. (c) Lateral negative steering. (d) Stationary rotation.

In addition, the high computational burden of the numerical optimization process can also affect the algorithm's real-time performance. Therefore, we aim to explore faster and more feasible alternative control solutions.

III. Method

To address the aforementioned issues, we propose a hierarchical trajectory tracking solution based on Deep Reinforcement Learning. Fig. 4 illustrates the overall structure of the proposed trajectory tracking method. The specific details of each component will be described in the following sections.

A. Low-level Controller

1) Soft Actor-Critic Algorithm: Soft Actor-Critic (SAC) [17] is a reinforcement learning algorithm designed for continuous action spaces and is capable of solving high-sample-complexity, model-free problems. SAC belongs to the Actor-Critic framework [18], which is also divided into Actor networks and Critic networks. Unlike traditional Actor-Critic frameworks, SAC's objective is not only to maximize the expected cumulative reward but also to maximize the entropy of each output action. The Critic network's output is represented by $Q(s_t, a_t)$, and the Actor network is represented by $\pi(a_t|s_t)$. $\hat{Q}(s_t, a_t)$ denotes the corresponding target action-value function. The loss function for the Critic network is defined as:

$$J_Q = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}\left(Q(s_t,a_t) - \hat{Q}(s_t,a_t)\right)^2\right]. \tag{10}$$

The loss function for the Actor network is defined as:

$$J_\pi = \mathbb{E}_{(s_t\sim D, a_t\sim\pi)}\left[\log\pi(a_t|s_t) - \frac{1}{\alpha}Q(s_t,a_t)\right]. \tag{11}$$

Fig. 5 shows our SAC network architecture. Since trajectory data have a temporal relationship, we introduce LSTM

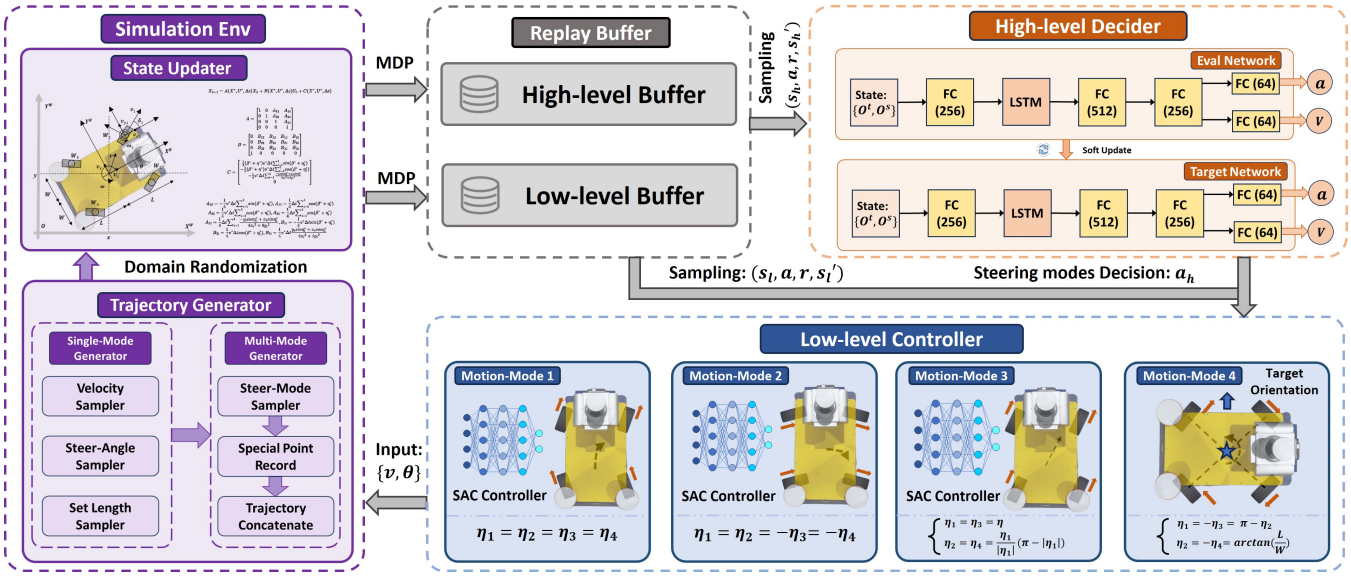


Fig. 4. Proposed hierarchical trajectory tracking framework based on deep reinforcement learning.

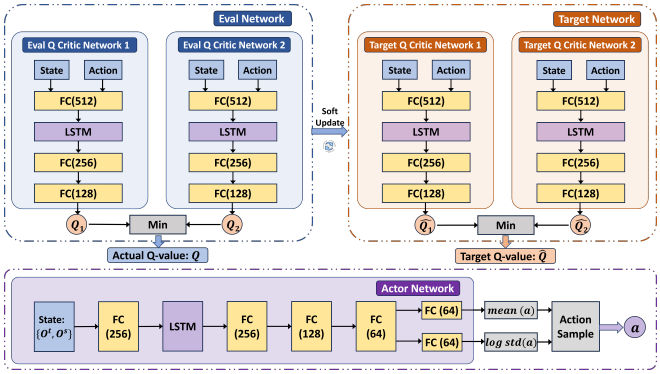


Fig. 5. Implementation framework of SAC network.

[19] into the network to enhance its ability to model sequential data. This allows the network to uncover potential long-term patterns, improving the model's prediction accuracy and robustness.

2) Observation Space: Our observation space $S_l = \{s_{tl}, s_{rl}\}$ consists of two parts: local reference trajectory information s_{tl} and robot state information s_{rl}.

To clearly represent the trajectory information ahead of the robot, we extract a segment of the trajectory and transform it into the robot's local coordinate system as the local reference trajectory. The coordinate transformation formula for the local reference trajectory is:

$$\begin{bmatrix} x_{\text{local}} \\ y_{\text{local}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & x_{\text{robot}} \\ -\sin(\theta) & \cos(\theta) & y_{\text{robot}} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{\text{global}} \\ y_{\text{global}} \\ 1 \end{bmatrix}, \quad (12)$$

where x_{global} and y_{global} are the coordinates of the trajectory in the global coordinate system before transformation, x_{local} and y_{local} are the coordinates in the local coordinate system after transformation, x_{robot} and y_{robot} are the coordinates of the robot in the global coordinate system, and θ is the robot's orientation angle in the global coordinate system.

The local reference trajectory information is defined starting from the reference point as:

$$s_{tl} = \{p_1, p_2, ..., p_n\}, \quad (13)$$

where n is the length of the reference trajectory, and $p_i = \{x_{\text{local}_i}, y_{\text{local}_i}\}, i = 0, 1, \ldots, n$.

The robot state information describes the robot's current motion status. We aim to use more generalized information when constructing the entire state space. Including the coordinates in the global coordinate system as part of the state space would reduce the generalization ability of the algorithm, so we define the robot state information as:

$$s_{rl} = \{\eta, v\}, \quad (14)$$

where v represents the robot's velocity.

3) Action Space: In designing the action space, we adopt an idea similar to MPC: using the system model to predict the state evolution over a period of time in the future. The action space is defined as:

$$A_l = \{A_1, A_2, ..., A_n\}, \quad (15)$$

where $A_i = \{a_i, \Delta\eta_i\}, i = 0, 1, \ldots, n$, a represents the robot's acceleration, and $\Delta\eta$ represents the wheel control angles of the robot.

We output a sequence of actions to enable the model to consider future steps while learning, and apply only the first action when the agent makes a state transition. This control method is also the core idea of MPC, which can effectively improve the robustness and stability of the control method.

4) Reward Function: In DRL, the reward function serves as feedback from the environment and plays a crucial role in training the optimal policy. We have designed a dense reward to evaluate the difference between the current robot state and the reference state, considering both accuracy and smoothness, to guide the agent in completing the trajectory tracking task. The specific reward type is:

$$R_l = w_1 R_s + w_2 R_a, \quad (16)$$

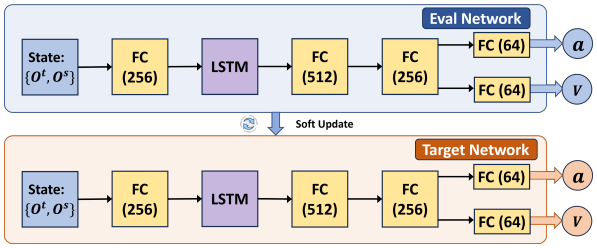


Fig. 6. Implementation framework of D3QN network.

where R_s and R_a represent the tracking accuracy reward and action reward, weighted by w_1 and w_2, respectively.

In the design of R_s, we aim to guide the robot to reach the target position on the trajectory while ensuring the overall optimality of the output action sequence to guarantee the foresight of the policy. We construct the reward function by using the negative Euclidean distance sum between the goal point sequence and the actual point sequence obtained by executing the output action sequence from the robot's current position. The reward function is:

$$R_s = -\sum_{i=1}^{n} \sqrt{(x_i - x_{t_i})^2 + (y_i - y_{t_i})^2}, \tag{17}$$

where x_i and y_i represent the robot's coordinates at the i-th step of the sequence, x_{t_i} and y_{t_i} represent the target coordinates at the i-th step of the sequence.

In the design of R_a, to ensure the smoothness of the trajectory and save energy, we want to penalize excessive variations in the control quantities. Therefore, we construct the action reward function as:

$$R_a = -\sum_{i=1}^{n} \mu^n (\Delta\bar{\eta}_j + \bar{a}_j), \tag{18}$$

where μ is a discount factor less than 1, and we want actions closer to the current point to have greater weight. To ensure uniformity across different modes, we need to normalize the action space to obtain $\bar{a}_j$ and $\bar{\eta}_j$, with the normalization function determined by the type of action space. In the scenario of this task, min-max normalization is sufficient to meet the needs of a linear action space. The action spaces after normalization are:

$$\{\bar{a}_j, \bar{\eta}_j\} = \left\{ \frac{a_j - a_j^{\min}}{a_j^{\max} - a_j^{\min}}, \frac{\eta_j - \eta_j^{\min}}{\eta_j^{\max} - \eta_j^{\min}} \right\}. \tag{19}$$

B. High-level Decider

1) Dueling Double Deep Q Network: Dueling Double Deep Q Network (D3QN) is a reinforcement learning algorithm designed for discrete action spaces, combining the ideas of Double DQN [20] and Dueling DQN [21]. D3QN network does not directly output the action-value function Q, but instead uses separate branches to output the state-value function V and the advantage function A. This approach allows for better predictions when states and values are relatively independent. The action-value function Q requires additional calculation by:

$$Q(s,a) = V(s,a) + A(s,a) - \overline{A(s,a)}, \tag{20}$$

where $\overline{A(s,a)}$ is the mean of the advantage function.

In addition, similar to Double DQN, D3QN eliminates estimation bias by using the evaluation network to determine the action, and the target network to determine the action value. ω_e and ω_t represent the evaluation and target network weights, the target value of D3QN is computed as:

$$y_t = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \omega_e); \omega_t). \tag{21}$$

The D3QN network architecture we use is shown in Fig. 6.

2) Observation Space and Action Space: Our observation space is quite similar to that of the low-level controller, $S_h = \{s_{th}, s_{rh}\}$, which is composed of two parts: the local reference trajectory information s_{th} and the robot state information s_{rh}.

Starting from the reference point, we define the local reference trajectory information as:

$$s_{th} = \{q_1, q_2, \ldots, q_n\}, \tag{22}$$

where $q_i = \{x_{\text{local}_i}, y_{\text{local}_i}, \theta_{\text{local}_i}\}$, with θ_{local_i} representing the orientation angle of the trajectory in the local coordinate system.

We define the robot state information as:

$$s_{rh} = \{M\}, \tag{23}$$

where M represents the robot's current motion mode. We use 0-3 to correspond to the four motion modes: symmetric negative steering, parallel positive steering, lateral negative steering, and stationary rotation.

In addition, we define the action space as:

$$A_h = \{N, T\}, N \in [1, n-1], T \in [0, 3]. \tag{24}$$

Action space indicates that a mode switch is required after step N. T refers to the steering mode selected after switching, and its value meaning is the same as that of M. After the decider concludes that a mode switch is required, the controller's observation space will no longer record the trajectory point after the switch point, but fill the void with the switch point so that the robot can stop at the switch point. After the switch is completed, continue reading the reference trajectory and tracking the trajectory.

3) Reward Function: We iterate backward by certain steps based on the selected motion mode and evaluate the effectiveness of the selected motion mode by utilizing the difference between the current robot state and the reference state. The reward function is:

$$R_s = \sum_{i=1}^{n} (-\Delta_{S_i}^T H \Delta_{S_i}), \tag{25}$$

where $-\Delta_{S_i} = [\Delta_{x_i}, \Delta_{y_i}, \Delta_{\theta_i}]$, with Δ_{x_i}, Δ_{y_i}, and Δ_{θ_i} representing the differences in x-coordinate, y-coordinate, and orientation angle between the actual state and the target state, respectively. H is a diagonal matrix that determines the coefficients for each state.

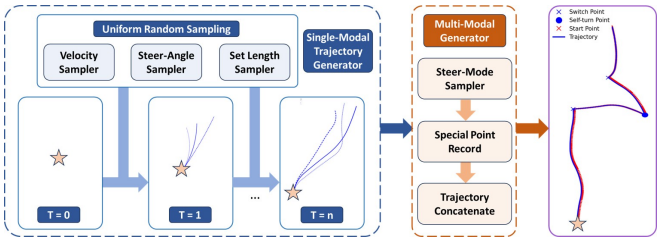


Fig. 7. Trajectory generator based on multiple randomization.

C. Reference Trajectory Generator

When generating the reference trajectory dataset required for algorithm training, we need to simulate the trajectory requirements in the real scene as much as possible. Our design not only considers the robot's kinematic properties but also randomizes key parameters such as speed, steering angle, and trajectory length during the generation process. Fig. 7 shows the workflow of our trajectory generator.

We first use a mode random sampler to determine the motion mode corresponding to the segmented trajectory to be generated. Then, based on the motion constraints of each motion mode and using multiple random samplers, we generate multiple segmented trajectories. Finally, these segmented trajectories are concatenated, and the attributes of the motion mode switch points are recorded.

Among them, because the trajectory does not specifically consider the robot's acceleration limit, errors will inevitably occur at some trajectory points during trajectory tracking. This situation corresponds to the scenario where robots receive unreasonable trajectories in actual scenarios, and the goal of the algorithm is to minimize errors as much as possible within the constraints of kinematics. The dataset of trajectories generated through multiple random samplings can effectively enhance the generalization ability of the algorithm and the reliability of its real-world deployment.

IV. Experiment

In this section, we will describe the parameter configurations and training conditions for the proposed trajectory tracking method. Additionally, several experimental scenarios have been designed to evaluate performance and validate the method's effectiveness. We use the PyTorch framework to build neural networks to obtain controllers and train them through reinforcement learning.

A. Experiment Setup

To validate our proposed trajectory tracking method, we conducted simulations and real robot experiments. The simulation experiments were performed on a computer equipped with an Intel Core i7-13700H CPU, a GeForce GTX 4060 GPU, and 32 GB of RAM. The software platform was configured with PyTorch 2.1.0 and CUDA 12.7. The overall reinforcement learning interaction framework is shown in Fig. 4, with key parameters listed in Table I. Additionally, we performed method performance verification on a real robot platform based on actual task requirements, its specific parameters are provided in Table II. To ensure consistency in the experiments, the key parameters of the numerical simulation model were set to match those of the real robot.

TABLE I

Reinforcement Learning Model Training Parameters

Parameter	Description	Value
M_{max}	Max memory size	100000
B	Batch size	256
n	Input sequence length	5
γ	Discount return factor	0.99
τ	Target smoothing coefficient	0.005
L_{r_1}	D3QN learning rate	0.002
ϵ	D3QN epslion-Greedy factor	[0.1,1]
L_{r_2}	SAC learning rate	0.0001
H	state coefficients matrix	diag(1,1,1)
ω_1, ω_2	controller reward coefficient	10, 0.02

TABLE II

Robot Related Parameters

Parameter	Description	Value
M, kg	Mass of the robot	250
W, mm	Width of the robot	710
L, mm	Length of the robot	1400
r, mm	Radius of the wheels	240
b, mm	Wheelbase of the robot	970
d, mm	Track of the robot	670
$a, m/s^2$	Max acceleration of the robot	5

B. Low-level Controller Experiment

To verify the performance of the low-level controllers corresponding to different motion modes, we compared the proposed method with traditional pure pursuit and the MPC trajectory tracking controller. We construct the environment of MPC controller through CXVPY [22] and use OSQP [23] to solve the problem. The tasks compared include random trajectories generated by the Reference Trajectory Generator under three different modes. Specifically, the time interval between every two trajectory points is 0.05s, and the generated trajectory velocity is between 0.5m/s-2m/s.

The steering angle constraints are defined as:

- Symmetric Negative Mode: $\eta \in [-\frac{\pi}{3}, \frac{\pi}{3}]$
- Lateral Negative Mode: $\eta \in [-\frac{\pi}{3}, -\frac{2\pi}{3}]$
- Parallel Positive Mode: $\eta \in [-\pi, \pi]$
- wheel steering rate limit $\Delta\eta \in [-\frac{\pi}{60}, \frac{\pi}{60}]$ per unit time

Fig. 8 demonstrates the experimental results of different trackers under three operational modes. We compared the three methods' performance in terms of lateral error, longitudinal error, response time, and other performance metrics. The lateral error is denoted as e_x, the longitudinal error as e_y, and the response time as Res Time. Experiments were conducted on 1000 randomly generated trajectories, each consisting of 100 trajectory points, and the maximum and mean errors were recorded. The specific results are shown in Table III.

Our method significantly outperforms pure pursuit algorithm in terms of tracking accuracy. Compared with the MPC approach, our method reduced the average lateral error by 47.53%, 34.59%, and 50.92% across three steering modes, respectively. In terms of longitudinal error, our method achieved average error reductions of 26.83%, 34.82%, and

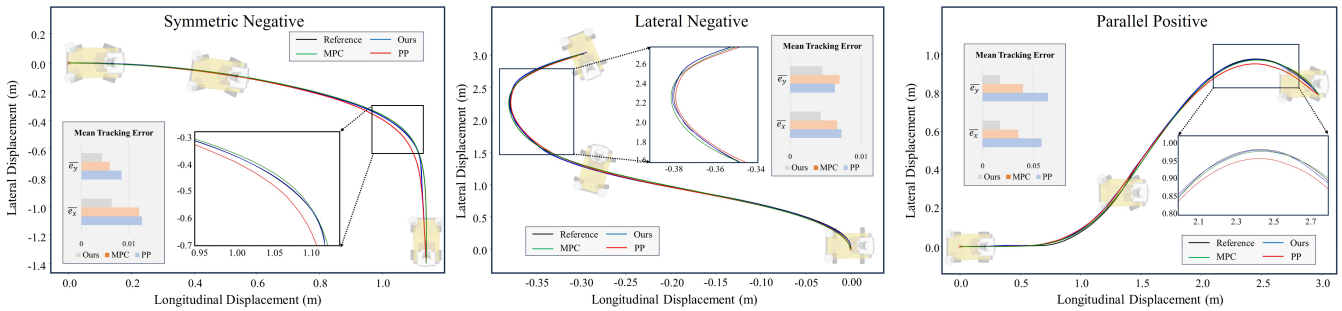


Fig. 8. Trajectory tracking results of Low-level Controller.

TABLE III

SINGLE-MODE TRAJECTORY TRACKING ERRORS AND COMPUTING TIME

Steer Mode	Method	Trajectory Following Metrics				
		$\|e_x\|$mean(m)	$\|e_x\|$max(m)	$\|e_y\|$mean(m)	$\|e_y\|$max(m)	Res Time(s)
Symmetric Negative	Pure Pursuit	0.01271	0.06358	0.008468	0.02679	0.000085
	MPC	0.01214	0.04116	0.005951	**0.01989**	0.032674
	Ours	**0.00637**	**0.02232**	**0.004354**	0.02038	0.001059
Lateral Negative	Pure Pursuit	0.00723	0.02871	0.006278	0.02934	0.000083
	MPC	0.00662	0.02627	0.006982	0.03282	0.032716
	Ours	**0.00433**	**0.02106**	**0.004551**	**0.02561**	0.001024
Parallel Positive	Pure Pursuit	0.05796	0.19091	0.064089	0.22522	0.000080
	MPC	0.03496	0.12895	0.039658	0.12934	0.031987
	Ours	**0.01716**	**0.10601**	**0.017154**	**0.07540**	0.001036

56.75%, respectively. Moreover, although the response time of our proposed algorithm is longer than that of the pure pursuit algorithm, it is reduced by 99% compared to the MPC approach, meeting the requirements for real-time deployment.

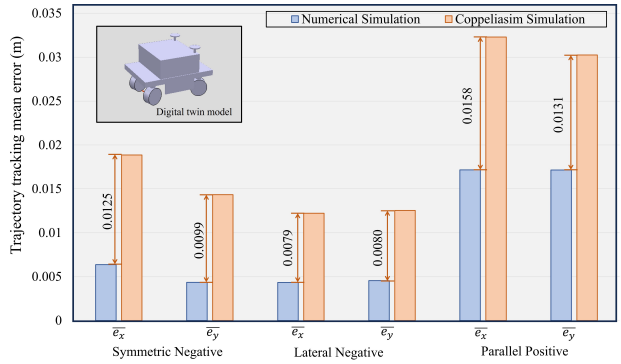


Fig. 9. Comparison of trajectory tracking results of low-Level controllers under numerical simulation and CoppeliaSim simulation.

To further validate the effectiveness of the algorithm, we created a digital twin model of a robot on the CoppeliaSim platform [24] based on the technical details of a real robot. Due to the independence of the high-level decider from motion models, we independently validated the performance of the low-level controller on the CoppeliaSim platform. The results in Fig. 9 show that the low-level controller's performance has slightly decreased compared to numerical simulation. However, it can still track the target trajectory stably.

C. High-level Decider Experiment

To validate the performance of the high-level decider in making mode-switching decisions, we used the random trajectories generated by the generator with various mode combinations for performance verification. The speed and motion constraints of each mode were consistent with the previous experiment. The low-level controller uses the proposed method. Fig. 10 shows the experimental results of the decider under a typical trajectory.

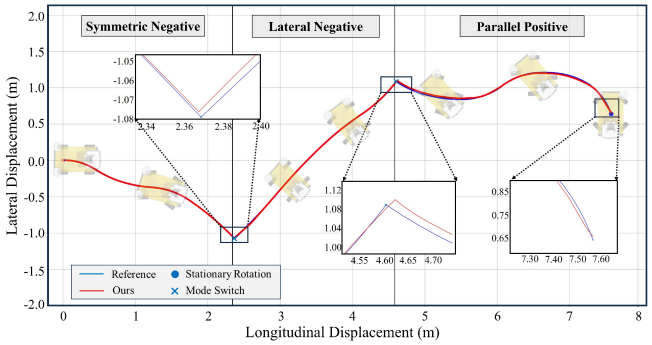


Fig. 10. Trajectory tracking results of High-level Decider.

We compared the position error between the decision points made by the decider and the actual switching points for each mode. Experiments were conducted on 1000 randomly generated trajectories composed of three motion modes and stationary rotation, followed by data statistics and the specific results are shown in Table IV. The vertical axis of the table is the current steering mode and the horizontal axis is the target steering mode.

TABLE IV

STEERING MODE SWITCH POINT ERRORS

Position error (m)	Symmetric Negative		Lateral Negative		Parallel Positive		Stationary rotation	
	mean	max	mean	max	mean	max	mean	max
Symmetric Negative	——	——	0.00491	0.02374	0.00486	0.00847	0.00482	0.00923
Lateral Negative	0.00766	0.03715	——	——	0.00941	0.03288	0.00791	0.03134
Parallel Positive	0.01842	0.04963	0.02077	0.05820	——	——	0.01952	0.06712

The steering mode switching error comes from the decision error, and the terminal error of the low-level controller. In all mode-switching scenarios, our decider consistently ensures high accuracy. A similar effect can be achieved by simulating all modes at each trajectory point and selecting the one with the smallest error, but our designed decider is significantly more time-efficient. Besides, due to the design of the action space, our designed decider avoids frequent mode switching.

D. Real-robot Experiment

Additionally, to validate the performance of the proposed method applied to a real robot, we deployed the algorithm on the robot mentioned earlier and tested it using the trajectory provided by an actual inspection task. The work scenario and inspection trajectory are shown in Fig. 11.

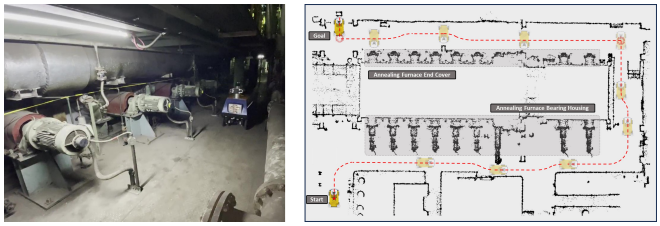


Fig. 11. Robot Inspection Environment and Inspection Path.

In our experiments, a collision risk is assumed whenever the distance between the robot and any obstacle falls below 2 cm, at which point tracking is considered to have failed. Methods relying on single steering mode were unable to successfully complete the tracking task. We conducted 20 tests using both the decider + MPC and decider + controller approaches respectively. The group use MPC achieved a tracking success rate of 75%, whereas our proposed method attained a success rate of 100%. These results confirm that our method can accurately track the predetermined inspection trajectory without any collisions with surrounding obstacles.

V. Conclusion

This paper presents a multi-modal trajectory tracking method considering the switch of steering modes. We enable the robot to autonomously choose the most suitable steering mode and perform stable trajectory tracking through the designed hierarchical control framework based on deep reinforcement learning and the corresponding target trajectory random generator and training interaction environment. The proposed algorithm has been successfully deployed on factory inspection robots, exhibiting excellent stability in actual production. However, this method is currently mainly verified based on a specific robot platform and a single environment, and has not yet fully considered the impact of actual factors such as positioning error on performance. Future work will focus on improving the robustness in more complex and dynamic environments, and explore the generalization ability of the method in multiple platforms and multiple scenarios to enhance its practical value and scope of application.

References

[1] L. F. P. Oliveira, A. P. Moreira, and M. F. Silva, "Advances in Agriculture Robotics: A State-of-the-Art Review and Challenges Ahead," *Robotics*, vol. 10, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2218-6581/10/2/52

[2] J. Guo, Y. Luo, and K. Li, "An Adaptive Hierarchical Trajectory Following Control Approach of Autonomous Four-Wheel Independent Drive Electric Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2482–2492, 2018.

[3] Z. Liu, Y. Xu, L. Zhang, S. Wang, and J. Wang, "The control strategy for vehicle transfer robots in ro/ro terminal environments," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[4] J. S. Keek, S. L. Loh, and S. H. Chong, "Comprehensive Development and Control of a Path-Trackable Mecanum-Wheeled Robot," *IEEE Access*, vol. 7, pp. 18 368–18 381, 2019.

[5] Y. Xu, Z. Chen, C. Deng, S. Wang, and J. Wang, "decision maker: Toward Dynamic Localization for Autonomous Landing of Unmanned Aerial Vehicle Based on LiDAR–Camera Fusion," *IEEE Sensors Journal*, vol. 24, no. 16, pp. 26 407–26 415, 2024.

[6] Y. Fu, X. He, S. Wang, and Y. Ma, "A Navigation Robot with Reconfigurable Chassis and Bionic Wheel," in *2004 IEEE International Conference on Robotics and Biomimetics*, 2004, pp. 485–489.

[7] Y. D. Setiawan, T. H. Nguyen, P. S. Pratama, H. K. Kim, and S. B. Kim, "Path tracking controller design of four wheel independent steering automatic guided vehicle," *International Journal of Control, Automation and Systems*, vol. 14, pp. 1550–1560, 2016.

[8] L. Tan, S. Yu, Y. Guo, and H. Chen, "Sliding-mode control of four wheel steering systems," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2017, pp. 1250–1255.

[9] V. Arvind, "Optimizing the turning radius of a vehicle using symmetric four wheel steering system," *International Journal of Scientific & Engineering Research*, vol. 4, no. 12, pp. 2177–2184, 2013.

[10] W. Lin, P. Wang, Y.-Y. Wu, W. Liu, and H.-J. Sun, "Reinforcement Learning-Based MPC for Tracking Control of 4WID4WIS," in *2023 2nd Conference on Fully Actuated System Theory and Applications (CFASTA)*, 2023, pp. 839–844.

[11] X. Liu, W. Wang, X. Li, F. Liu, Z. He, Y. Yao, H. Ruan, and T. Zhang, "MPC-based high-speed trajectory tracking for 4WIS robot," *ISA Transactions*, vol. 123, pp. 413–424, 2022.

[12] N. T. Nguyen, P. Tej Gangavarapu, N. Mandel, R. Bruder, and F. Ernst, "Motion planning for 4WS vehicle with autonomous selection of steering modes via an MIQP-MPC controller," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 9765–9771.

[13] F. Xu, X. Liu, W. Chen, and C. Zhou, "Dynamic Switch Control of Steering Modes for Four Wheel Independent Steering Rescue Vehicle," *IEEE Access*, vol. 7, pp. 135 595–135 605, 2019.

[14] Y. Wang, D. Li, X. Zhuang, Y. Wang, S. Yang, and R. Wu, "An adaptive path tracking controller for autonomous vehicles based on the pure pursuit algorithm," in *2023 China Automation Congress (CAC)*, 2023, pp. 7508–7512.

[15] G. Zhao, Z. Chen, and W. Liao, "Reinforcement-tracking: An end-to-end trajectory tracking method based on self-attention mechanism," *International Journal of Automotive Technology*, vol. 25, no. 3, 2024.

[16] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[18] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[19] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W. kin Wong, and W. chun Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," 2015.

[20] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[21] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[22] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[23] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

[24] E. Rohmer, S. P. N. Singh, and M. Freese, "CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.