Reinforcement-Tracking: an end-to-end trajectory tracking method based on self-attention mechanism

Weiming Liao[1,1], Guanglei Zhao[1,1*†] and Zihao Chen[1,1†]

[1*]The Insititute of Electrical Engineering, Yanshan University, Hebei Street West, Qinhuangdao, 066004, Hebei, China.

*Corresponding author(s). E-mail(s): zglmj@ysu.edu.cn;
Contributing authors: mingx@stumail.ysu.edu.cn;
476379292@qq.com;
†These authors contributed equally to this work.

Abstract

As a bridge between planners and actuators, trajectory tracking is a critical and essential part of robot navigation, autonomous driving and other fields. While traditional trajectory tracking methods generally have disadvantages such as poor tracking accuracy, high modeling requirements, and heavy computational load.This paper proposes an end-to-end trajectory tracking method based on reinforcement learning, and an information encoding network and a reinforcement learning policy network are constructed. A multi-task dense reward function for trajectory tracking is designed.For efficient encoding of local trajectory information, a self-attention mechanism is introduced.A virtual simulation environment is constructed for model training by modeling the trajectory tracking task.Comparing the proposed method with model predictive control and pure pursuit in the tracking experiments of several reference trajectories. The results show that our proposed method has significant advantages in terms of lateral tracking.

Keywords: Reinforcement learning, trajecotry tracking, self-attention

1 Introduction

The autonomous movement and navigation capabilities of unmanned ground vehicles (UGVs) have greatly improved due to rapid advancements in computer technology and artificial intelligence in recent years. And among the most popular and reliable approaches to achieving autonomous navigation is the perception-planning-control navigation pipeline[1]. This pipeline has been widely adopted in various autonomous systems, such as service robots, self-driving cars, and unmanned aerial vehicles, to accomplish navigation tasks. The trajectory tracking control, serving as an intermediate bridge between the planning and execution layers within this approach, plays a pivotal role in determining the effectiveness and overall performance of the autonomous navigation system.

In practical trajectory tracking control, several factors can affect the performance of path tracking, including the nonholonomic constraints of the unmanned vehicle. The constraints of the kinematic and dynamic models can impact the trajectory tracking performance in real control situations. Additionally, the actuator performance constraints, such as the unmanned vehicle's speed and minimum turning radius can limit system performance. Different control strategies are necessary to meet these intrinsic requirements to accommodate various constraints. Situational constraints also play a critical role in trajectory tracking. For instance, in autonomous driving situations, passenger comfort and lane centerline maintenance are important considerations.

To summarize, different trajectory-tracking control methods are necessary to accommodate distinct control requirements. Preview-based tracking control approaches can produce effective results for trajectory tracking with moderate speed and less extreme curvature changes. In contrast, optimization-based trajectory tracking methods can handle high dynamic scenarios but may have drawbacks, such as design complexity and relatively long response time, which limit their widespread use. Moreover, the performance of the controller depends on the accuracy of the model.

Reinforcement learning (RL) has made significant advances in recent years due to the progress made in deep learning and hardware technology. This progress has led to the increasing utilization of RL in academic settings to address control and decision-making issues[2, 3]. The utilization of RL has simplified controller design and provided superior control outcomes with sufficient appropriate samples. Moreover, it enables the implementation of sophisticated and highly dynamic trajectory tracking control, such as turning operations on roads[4].

In light of the aforementioned challenges, this paper presents a new end-to-end trajectory tracking method based on Dueling Double Deep Q-networks(DDQN)to achieve effective control of complex motion trajectories. Rather than using global reference trajectory information, our approach employs local reference trajectory as input for trajectory tracking. When processing reference trajectory data, a self-attention mechanism is introduced to extract useful information from trajectory segments, resulting in a smaller

model and a quicker response speed. Regarding the reward strategy, we employ a dense rewards policy to enable efficient network training. Moreover, we introduce a Lattice Graph method in the action space extraction to achieve better discrete control and reduce the training time[5].

Through experimental comparison, our designed controller has higher control frequency compared to the optimization-based controller. Meanwhile, our controller delivers better control performance than the preview-based approach.

2 Related works

Preview-Based trajectory tracking controllers

The common feature of the preview-based control methods is that a point on the reference trajectory is selected as the preview point, and the control output is calculated based on the relative relationship between the preview point and the unmanned vehicle.The most widely-used and straightforward method for trajectory tracking is the Proportional-Integral-Derivative (PID) based tracking control method, which often involves decoupling the control of the unmanned vehicle into horizontal and vertical components to achieve effective trajectory tracking. Fractional Order PID (FOPID) is utilized for path tracking control and effectively reduces the overshoot of the system, despite the unmanned vehicle trajectory tracking control having a high degree of nonlinear and dynamic features.[6]. In order to overcome the shortcomings of the lack of a system model for PID, geometric model-based lateral tracking methods have been proposed. Common methods include pure pursuit control, which calculates the vehicle control angle based on the geometric relationship, essentially converting the lateral error between the vehicle location and the desired trajectory at the pre-sighting point into a controlled quantity, thus reducing the deviation[7]. Stanley's method takes the front wheel of the unmanned vehicle as the center and takes into account the lateral error and the deviation of heading angle, and realizes path tracking by the control of the front wheel turning angle.[8]. The benefit of the above control methods is that complex models are not required. While these methods fail to take into account the dynamics of the vehicle, as a result, the steady-state error of the control system will increase as the speed increases, which makes it difficult to cope with high-speed motion scenarios.

Optimization-Based trajectory tracking controllers

Aiming at the issues of preview-based approaches, the linear quadratic regulator(LQR) tracking control based on the optimal control theory is developed, and the lateral controller is designed by establishing the vehicle dynamics model, which greatly improves the tracking performance. However, the control of real vehicles often has various constraints, and the LQR control is difficult to handle the constraints flexibly. Meanwhile, the algorithm is for whole time domain control, and the anti-interference ability is not strong. When the curvature varies quickly, there is a significant overshoot, which can easily

lead to system instability[9]. In order to overcome this series of issues, Trajectory tracking control based on model predictive control(MPC) has been proposed. MPC uses the predictive model of the system to predict the future state and determines the control sequence by the receding optimization while taking into account various constraints. Feedback correction is used to continuously adjust the control output at the sampling moment to achieve finite time-domain control of the system.[10]. Whereas it also creates new problems because receding optimization increases the algorithm's time complexity, places heavy demands on the hardware, and causes the calculation time to grow exponentially with model complexity. In the article [11], The authors embed the PID speed controller into the model predictive control controller to reduce the computational effort of solving the optimization problem and achieve the tracking of the continuous and fast-changing trajectory of curvature. However, MPC is a model-based control algorithm, and the model limits the control performance. In complex scenarios, how to construct an accurate model directly affects its control performance, which greatly limits the design and generality of the controller.

Deep learning methods

In order to reduce the complexity of the controller, many researchers want to implement end-to-end navigation systems by using sensors as input and neural networks to learn navigation or path tracking. It can achieve optimal performance with sufficient training samples[12]. In study [13], the authors utilized image inputs from dual cameras and a 6-layer convolutional neural network to establish the mapping between the images and the turning angle of the car to achieve end-to-end lateral trajectory tracking control. Despite the effectiveness of deep learning techniques, they are constrained by their limited generalization capability, which limits their applicability in real-world systems. In order to enhance the performance of control systems, several researchers have merged neural networks with conventional feedback feedforward controllers. For instance, Spielberg et al. [14] utilized neural networks and feedback control to design controllers and evaluate their performance in vehicle control. The study's findings revealed that this integrated approach outperforms the control of professional drivers, demonstrating that combining deep learning methods with conventional control methods could be an effective strategy to enhance the performance and stability of control systems.

Reinforce learning methods

Reinforcement learning is an emerging approach to solving robot control challenges [15], which learns optimal control strategies to maximize periodic cumulative rewards through the interaction of intelligence with its environment. Deep reinforcement learning employs neural networks to approximate state value functions or policies, enabling it to adapt to complex environmental changes and effectively handle nonlinear and high-dimensional control problems. As a result, this approach has been widely used in the field of complex robot control[16, 17]. In the task of trajectory tracking control, the authors propose trajectory tracking control methods based on Deep Deterministic

Policy Gradient(DDPG) [18], which incorporates Actor-critic algorithms [19], deterministic policy gradient [20], DQN algorithm [21] to implement the path tracking control of vehicles in TORCS simulation. Nevertheless, the DDPG algorithm may encounter issues such as overestimation of state action values, large cumulative errors, and low training efficiency [22]. To tackle these challenges, certain researchers have integrated a dual evaluation network and prioritized experience replay mechanism to enhance the performance of the DDPG algorithm and facilitate its application in the lane holding task of autonomous vehicles[23]. Liu et al. use point cloud data and satellite maps to extract real reference road information and generate reference trajectories. They further enhanced the trajectory tracking performance by integrating imitation learning and reinforcement learning, which resulted in accurate trajectory tracking control for urban roads[24]. Nevertheless, while reinforcement learning algorithms tend to perform well in simulations, deploying them in real-world applications often requires substantial effort [25].

3 Trajectory tracking process and environment

3.1 Markov decision process(MDP)

The Markov Decision Process (MDP) serves as the fundamental framework for reinforcement learning, enabling the representation of agent-environment interactions. MDP embodies the Markov property, which stipulates that the conditional probability distribution of a random process's future state, given the current state and all previous conditions, is solely determined by the present state. Consequently, the actions of an agent in the current state exert an influence on both present and future returns[26]. Let the sequence of random variables with Markov property be $s_1, s_2, \cdots, s_t$,which satisfies:

$$p\left(s_{t+1} \mid s_t\right) = p\left(s_{t+1} \mid s_1, s_2, \cdots, s_t\right) \tag{1}$$

In the Markov process, the introduction of a reward function results in the formation of a Markov reward process (MRP).In order to distinguish the importance of the reward, the discount factor $\gamma$is introduced, and the cumulative discount return after time t is as follows:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots + \gamma^{T-t-1} r_T \tag{2}$$

Where,$T$is the final moment, and $r(t)$is the reward at the moment t

The state value function V(s) in the process of MRP is calculated by expectation and expressed as:

$$\begin{aligned} V(s) &= \mathbb{E}\left[G_t \mid s_t = s\right] \\ &= \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T \mid s_t = s\right] \end{aligned} \tag{3}$$

Through the Bellman Equation derived [27], $V(s)$ can be represented as:

$$V(s) = R(s) + \gamma \sum_{s' \in S} p\left(s' \mid s\right) V\left(s'\right) \tag{4}$$

Where s' denotes a future state, $p(s'mids)$refers to the probability of transferring from the current state to the future state,$V(s')$ represents the value of a future state, and $R(s)$ represents the immediate return. $\gamma \sum_{s' \in S} p\left(s' \mid s\right) V\left(s'\right)$ represents totally discount return in the future .

3.2 Reinforcement learning process background

We assume a reinforcement learning process defined by $\{S_t, A_t, R_t, \pi(s,a), p(s_{t+1}, s_t)\}$. Here, S_t, A_t, and R_t denote the environmental state observed by the agent at time t, the action taken by the agent, and the associated reward, respectively. The behavior strategy of the agent is represented by $\pi(s,a)$, where $\pi(a \mid s) = p(A_t = a \mid S_t = s)$. To evaluate the merits and drawbacks of a given strategy, the action-value function $Q_\pi(s,a)$ is introduced to represent the cumulative discount return of the action $a$taken according to the strategy πunder the state s, which is expressed as:

$$Q_\pi(s,a) = E_\pi(G_t \mid S_t = s, A_t = a) \tag{5}$$

The derivation from the Bellman expectation equation yields $V(s)$

$$V_\pi(s) = \mathbb{E}_\pi \left[r_{t+1} + \gamma V_\pi\left(s_{t+1}\right) \mid s_t = s\right] \tag{6}$$

For the Q function, there are:

$$Q_\pi(s,a) = E_\pi(R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a) \tag{7}$$

The optimal action value function $Q_*(s,a)$ can be obtained through the Bellman optimal equation:

$$Q_*(s,a) = \max_a Q_\pi(s,a) \tag{8}$$

which denotes the maximum expected value under the optimal policy π^* when the agent is in state s and takes action a and the optimal state value function is defined as

$$V^*(s) = \max_\pi V_\pi(s) \tag{9}$$

By associating $V(s)$ with the Q function [28], we obtain the advantage function:

$$A_\pi = Q_\pi(s,a) - V_\pi(s) \tag{10}$$

3.2.1 Dueling network DDQN

Neural networks have been leveraged to model the value function to enhance the robustness of reinforcement learning and address the issue of unknown states. By combining deep learning methods' environment comprehension capabilities with reinforcement learning methods' decision-making proficiency, neural networks provide an effective solution. One such approach is the Deep Q-Networks (DQN) algorithm, proposed by Mnih et al. [21]. DQN leverages experience replay to break the correlation among sampled data, thereby rendering the data distribution more stable and improving data utilization efficiency [21]. In DQN, the neural network approximation function using $Q(s, a;\ w)$ for $Q^*(s, a)$, w is the neural network parameter. For instance, the Loss function for the ith step iteration is:

$$L_i(w_i) = E[(y_i - Q(s, a;\ w_i))^2] y_i = r + \gamma \max_{a'} Q\left(s', a';\ w^-\right) \tag{11}$$

where w^- is the target network parameter, and a' denotes all possible actions. Although DQN has achieved some success in some aspects, its intelligent body behavior selection and Q-value calculation use the same network, which can cause overestimation problems. To make the target Q-value estimation more accurate, DDQN mitigates the overestimation problem to some extent by changing the way Q-values are computed [29]. Further, the Dueling DQN method optimizes the evaluation of action value by adding a dueling network structure to the original DQN algorithm. It treats the Q function into two parts, the value function and advantage function, which makes the Q function network converge better [28]. Furthermore, the advantage function is centralized to improve identifiability. In this paper, we combine the advantages of Dueling DQN and DDQN to build Dueling DDQN. The Q-function expression is as follows:

$$Q(s, a;\ w, \alpha, \beta) = V(s;\ w, \beta) + \left(A(s, a;\ w, \alpha) - \frac{1}{|A|} \sum_{a'} A\left(s, a';\ w, \alpha\right) \right) \tag{12}$$

where $|A|$ denotes the number of selectable actions. α and β are the parameters denoting the value function network and the advantage function network, respectively. As the importance of different training samples may vary during the learning process, using the Priority Replay Buffer approach to accelerate network training[30]. And the network is updated using the following equation:

$$L_i\left(w_i\right) = E\left[\left(r + \gamma Q\left(s', \operatorname{argmax} Q\left(s', a;\ w_i\right);\ w_i^-\right) - Q\left(s', a;\ w_i\right)\right)^2\right] \tag{13}$$

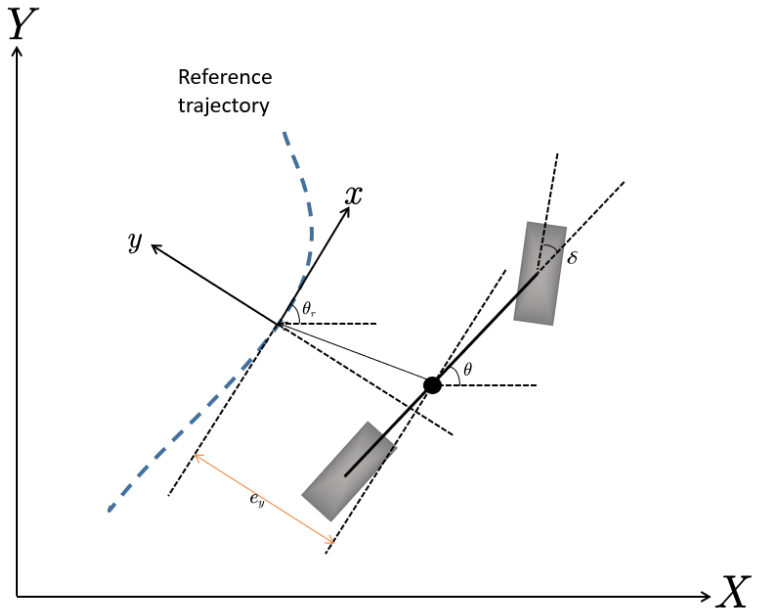


Fig. 1: UGV Tracking Model

3.3 Reinforcement learning-based trajectory tracking modeling

3.3.1 State and action space setting

As shown in Figure.1, consider the position of the unmanned vehicle in the global coordinate system as (x, y), with a velocity vector of $v = (v_x, v_y)$. Let the matching point of the unmanned vehicle on the reference trajectory be denoted as (x_r, y_r), the heading angle and angular velocity as θ and $\dot{\theta}$. The tangential angle of the matching point as θ_r. And the lateral error with the matching point as the reference coordinate system can be expressed as e_y:

$$e_y = \begin{bmatrix} \cos(\theta_r) & -\sin(\theta_r) \\ \sin(\theta_r) & \cos(\theta_r) \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} \tag{14}$$

where $(dx, dy)^T = (x_r - x, y_r - y)^T$; The angular error is defined as$e_\theta = \theta_r - \theta, e_\theta \in [-\pi, \pi]$,

Using the matching point as the starting point, define the local reference trajectory as

$$B = \{b_1, b_2, \cdots, b_n\} \tag{15}$$

where n is the maximum index value of the local reference trajectory,$b_i = \{x_{ri}, y_{ri}, \theta_{ri}\}, i = 0, 1, \cdots, n$, The observable state S of the trajectory tacking

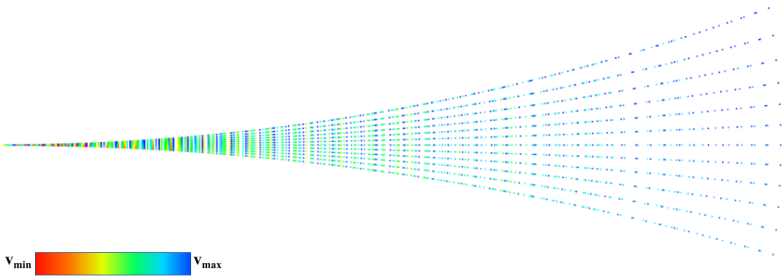


Fig. 2: Action space sampling

task can be defined as :

$$S = \{x, y, v, e_y, e_\theta, \theta, \dot{\theta}, B\} \tag{16}$$

In the action space setting, we introduce the lattice graph method into the action space discretization and construct a single-layer trajectory library, which conforms to the kinematic model (17)

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = v\frac{\tan\delta}{L} \end{cases} \tag{17}$$

Where L is the wheelbase. Table 1 displays the discrete parameters of the trajectory library. In an effort to streamline the number of action spaces, we perform an analysis to identify overlaps within the single-level trajectory library. Specifically, if the degree of overlap between two trajectories exceeds a preset threshold, we eliminate the overlapping trajectories. The resulting trajectory library is illustrated in Fig. 2.

Table 1: Discrete parameter

Parameters	value
Velocity range(m/s)	[0,1]
Velocity interval(m/s)	0.1
Steering angle range (deg)	[-33,33]
Steering interval(deg)	0.015
Discrete time(s)	0.1

The set of discrete actions is denoted as :

$$A = \{g_1, g_2, \cdots, g_m\} \tag{18}$$

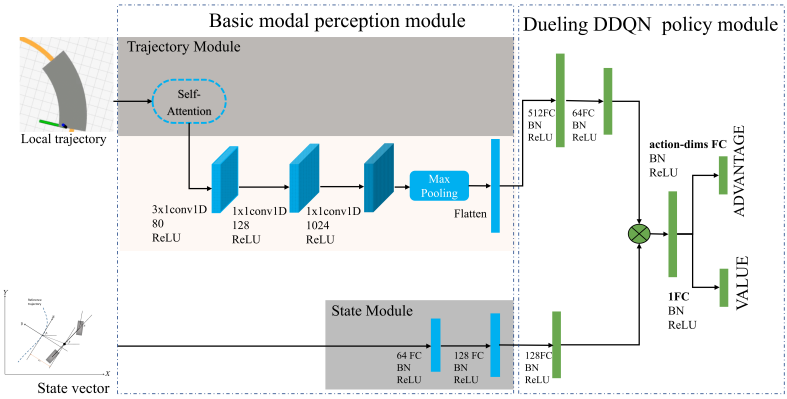


Fig. 3: Network structure based on the Dueling DDQN framwork

where m is the action size. $g_i = v_i, \delta_i$, $v_i \in [v_{min}, v_{max}]$ denotes the speed of unmanned vehicle , and $\delta \in [\delta_{min}, \delta_{max}]$ denotes its front wheel turning angle.

3.3.2 Network structure design

According to the state space and action space settings of the trajectory tracking model, the proposed reinforcement learning trajectory tracking network is shown in figure 3. The network consists of a front-end perception module and a Dueling DDQN policy network module.

Since the state of the vehicle itself and the information of the reference trajectory are considered in the state space, the front-end perception module is comprised of two fundamental network coding modules: the reference trajectory information encoding module and the posture information encoding module. We employ the trajectory encoding module to process the local trajectory information to be tracked, which discretizes the trajectory into individual points so that it can be input into the network in the form of point clouds. Subsequently, three Multi-Layer Perception (MLP) layers and a maximum pooling layer are used for feature extraction. The posture information encoding module utilizes a linear layer for feature extraction, with a focus on analyzing the current posture, action state, and relative posture state of an unmanned vehicle with respect to the trajectory reference point. The DQN policy network module first decodes the high-dimensional information encoded by the two encoding networks and integrates them into a high-dimensional feature vector. Subsequently, the Dueling DDQN network model is employed, whereby the aforementioned high-dimensional vector is fed into both the advantage network and the value network. Notably, according to Eq. 12, we let the advantage network and the value network share some network parameters.

3.3.3 Self-Attention module

The reference trajectory encoding module faces a two-fold challenge. First, the input local reference trajectories contain a considerable amount of high-dimensional, highly correlated information. Therefore, it is crucial to accurately extract the features of the trajectories and distinguish the significance of trajectory points b_i over the entire local trajectory B. Second, reference trajectories in local trajectory tracking may vary in length, resulting in variable network inputs. To address these challenge, this paper utilizes the Multi-Scale attention (MUSE) mechanism to process local trajectories of different shapes or multiple scales[31]. The MUSE is adept at identifying the dependencies between local reference trajectories, enhancing the network's robustness.

3.3.4 Reward function

Since trajectory tracking measures the tracking performance of each slight segment trajectory, we design dense rewards to evaluate the difference between the current unmanned vehicle state and the reference state in order to guide the agent through the trajectory tracking task, with the following types of rewards:

horizontal reward: The horizontal reward is applied to limit the lateral distance of the unmanned vehicle from the reference trajectory, which sets a segmented linear reward function based on the absolute value of the horizontal distance from the matching point to the vehicle:

$$r_y = \begin{cases} \mu_y, & \mid e_y \mid > \mid \widetilde{e_y} \mid \\ \kappa_y^i \mid e_y \mid + \xi_y^i, & \mid e_y \mid < \mid \widetilde{e_y} \mid, \mid e_y \mid \in \left[d_y^i, d_y^{i+1}\right] \\ 0, & \mid e_y \mid > d_y^3 \end{cases} \tag{19}$$

Where μ_y is the lateral penalty coefficient. e_y, $\widetilde{e_y}$ denote the lateral error at the current and previous moment, respectively. Furthermore, $\left[d_y^i, d_y^{i+1}\right], i = 0, 1, 2$ denote the different segmented lateral error intervals, and κ_y^i, ξ_y^i are the reward coefficients and biases of the segmented curves in the corresponding transverse error intervals.

Hearding reward: The heading reward corrects the heading of the unmanned vehicle with respect to the reference trajectory, which is also described by a segmented linear function.

$$r_a = \begin{cases} \mu_a, & \mid e_a \mid > \mid \widetilde{e_a} \mid \\ \kappa_a^i \mid e_a \mid + \xi_a^i, & \mid e_a \mid < \mid \widetilde{e_a} \mid, \mid e_a \mid \in \left[d_a^i, d_a^{i+1}\right] \\ -1, & e_a > d_a^1 \end{cases} \tag{20}$$

Where μ_a denotes the heading penalty coefficient, e_a, $\widetilde{e_a}$ denotes the heading error between the current and the previous moment, respectively. $\left[d_a^i, d_a^{i+1}\right], i = 0, 1$ denote the different segmented heading error intervals

respectively. κ_a^i, ξ_a^i denote the reward coefficients and biases of the segment curves corresponding to the heading error intervals.

Angular velocity reward: In order to reduce the oscillation of the unmanned vehicle, this paper applies a negative reward to the vehicle's excessive angular velocity change rate.

$$r_{\dot{\theta}} = \begin{cases} 1, & \dot{\theta} \leq d_{\dot{\theta}}^1 \\ \kappa_\omega^1 \mid \dot{\theta} \mid, & \dot{\theta} \in \left[d_{\dot{\theta}}^1, d_{\dot{\theta}}^2\right] \\ \kappa_\omega^2 \mid \dot{\theta} \mid, & \dot{\theta} \geq d_{\dot{\theta}}^2 \end{cases}, \tag{21}$$

where $\dot{\theta}$ is the angular velocity. $\left[d_{\dot{\theta}}^i, d_{\dot{\theta}}^{i+1}\right], i = 0, 1$ denote the angular velocity segment interval and κ_ω^i denotes the angular velocity penalty factor of the i_{th} segment

longitudinal reward: The longitudinal reward is used to motivate the unmanned vehicle to move forward along the reference trajectory, which is given according to the index of matching points on the reference trajectory corresponding to the position of the unmanned vehicle.

$$r_l = k_l(l - \widehat{l})/N \tag{22}$$

Where N denotes the index corresponding to the endpoint of the global trajectory. l and $\widehat{l}$ denote the index of the reference trajectory corresponding to the current moment and the previous moment, respectively, k_l is a proportional factor.

endpoint reward: Give positive reward when the unmanned car reaches near the end of the trajectory M:

$$r_g = M \tag{23}$$

The reward vector, denoted as $r = [r_y, r_a, r_l, r_{\dot{\theta}}, r_g]$, is a combination of rewards. To customize track tracking, different weights are assigned to various types of rewards. The weight vector, represented as $\Lambda = \lambda_y, \lambda_a, \lambda_l, \lambda_{\dot{\theta}}, \lambda_g$, reflects the weights assigned to each type of reward. Consequently, the total reward function can be expressed as $r_{total} = r \cdot \Lambda^T$.

4 Experiments and evaluation

In this section, we describe the implementation of the proposed reinforcement learning method for the trajectory tracking task and evaluate its performance. To accomplish this, we create a reinforcement learning environment using the ROS system and Gazebo simulator[32]. Furthermore, we construct a corresponding neural network using PyTorch and obtained the tracking controller through reinforcement learning model training.

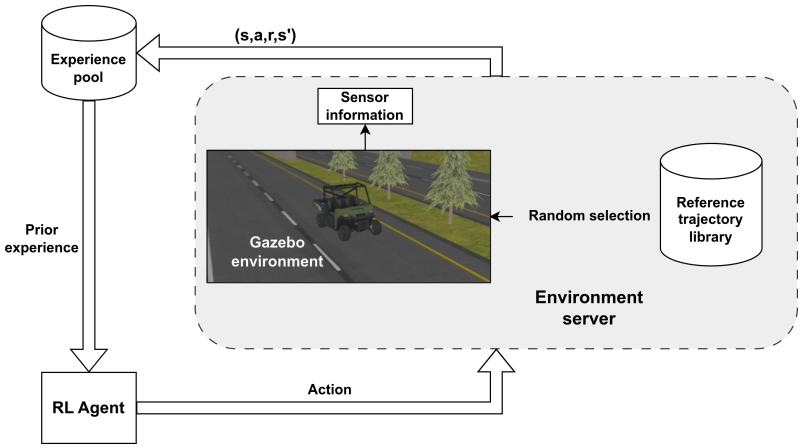


Fig. 4: Reinforcement learning process of unmanned vehicle trajectory tracking

4.1 Experiment environment

To implement our proposed reinforcement learning approach, we utilize a high-performance computer equipped with an Intel Core i7-10875h CPU, a GeForce GTX3060 GPU, and 32 GB of RAM. The PC is configured with Ubuntu 20.04, ROS Noetic, Pytorch 1.10.2, and Python 3.8. To accurately simulate the Ackermann structural model, we construct a simulation model using the parameters shown in Table 2. This virtual Model's kinematic and dynamic properties closely resemble those of a real vehicle. After that, we design an interaction model for reinforcement learning based on this simulation model, as depicted in Fig. 4.

Table 2: Model parameters

Parameters	value
Wheelbase(m)	2.054
Vehicle width(m)	1.43124
Mass (m/kg)	720
Steering angle range (deg)	[-33,33]
Velocity range(m/s)	[0,1]
Max acceleration(m/s^2)	1.0
Max steering angle velocity(deg/s)	10.0

4.2 Reinforcement learning model training

We train the reinforcement learning network proposed in subsection 3.3.2 using the interaction model outlined above. In order to ensure that the unmanned

vehicle can obtain valid information during the training process, we set the following environment reset conditions:

1. When the lateral distance of the unmanned vehicle from the reference trajectory is greater than a certain predefined threshold $\mathcal{L}_{th}$
2. When the total number of steps interacted by the unmanned vehicle in the episode is greater than a certain predefined threshold $\mathcal{N}_{th}$
3. When the Euclidean distance of the unmanned vehicle from the end of the trajectory is less than a certain predefined threshold $\mathcal{D}_{th}$

As a result, the training parameters of the reinforcement learning-based trajectory tracking model are shown in Table 3:

Table 3: Reinforcement learning model training parameters

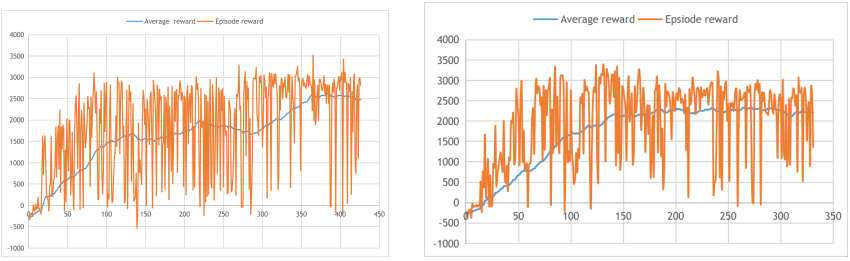
Parameters	Description	Value
L_r	Learning rate	0.001
$\mathcal{M}_{max}$	Max memory size	5000
$\mathcal{B}$	Batch size	360
γ	Discount return factor	0.9
$\mathcal{L}_{th}$	Lateral distance threshold	3.0
$\mathcal{N}_{th}$	Maximum number of interactive steps per episode	1000
$\mathcal{D}_{th}$	Distance threshold from the end of the target trajectory	1.0

During the training process, the interaction data of the unmanned vehicle, such as observation status, action and reward, are collected and stored in the experience pool. A batch of data is sampled from the experience pool by priority sampling for model parameter update until the episode termination condition is met and the environment restarts. To enhance the model's generalization capability, we randomize the reference trajectory and the position of the unmanned vehicle relative to the starting point of the trajectory each time the episode is restarted.

Figure 5 shows the average reward return and the change of return per round during the training process. The figure shows that the algorithm achieves convergence in about 150 episodes with the incorporation of self-attention mechanism. In contrast, the network without the self-attention mechanism converged in roughly 370 episodes. This indicates that the self-attention mechanism can extract useful information from local trajectories and has a remarkable fast learning ability.

4.3 Simulation comparison experiment

To validate the effectiveness of the reinforcement learning trajectory tracking model, our proposed method is compared with the conventional pure pursuit and MPC trajectory tracking controller in the trajectory tracking tasks of the preset trajectory and the planning trajectory.



(a) (b)

Fig. 5: Reward curve of learning porcess. (a) training without self-attention,(b) training with self-attention

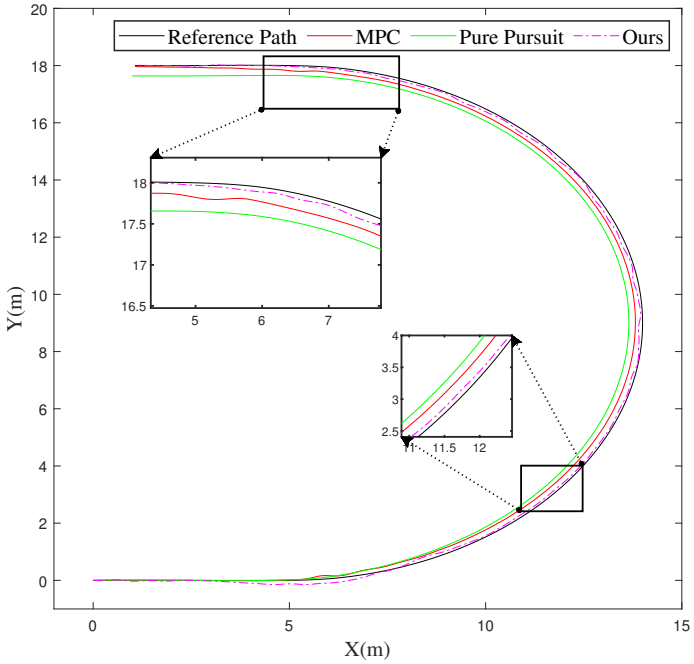


Fig. 6: Effect of preset trajectory tracking

4.3.1 Preset trajectory tracking

In the tracking task of the preset trajectory, we designed a U-shaped trajectory to observe the tracking performance of three different controllers.

Figure 6 shows the qualitative effects of the three trajectory tracking. For quantitative analysis, we compare the performance metrics of 3 methods

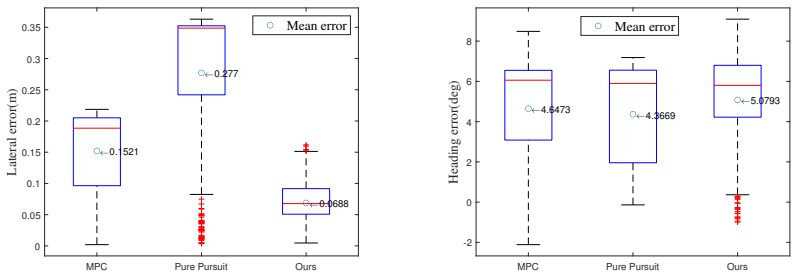
regarding the lateral error, heading error, response time, and trajectory tracking time. As shown in the table 4, the response time of the proposed algorithm is similar to that of the pure pursuit algorithm and much smaller than that of the MPC algorithm.

Regarding trajectory tracking time, it can be seen from figure.6 that the proposed algorithm travels a path length greater than the two algorithms, which is basically equal.

Table 4: Performance analysis in preset trajectory tracking

Method	Lateral error(m)	Heading error(deg)	Response time(ms)			Travel time(s)
	Mean	Mean	Min	Max	Mean	Mean
Ours	**0.07**	5.08	2.33	6.63	2.84	39.45
Pure pursuit	0.28	**4.37**	0.04	0.05	0.15	36.14
MPC	0.15	4.65	32.50	120.01	43.97	36.31

As shown in Figure 7b,our method has similar mean values as the other methods in terms of heading error. However, it has a smaller error distribution, which indicates that our method is more stable in the control of the heading. For the lateral error shown in Fig. 7a, the average value of the lateral error and the distribution of lateral error in the range of $25\% - 75\%$ of the proposed method are smaller than those of MPC and pure pursuit methods, with small fluctuations and stable lateral error. This indicates that our method has apparent advantages in tracking control of the lateral error. Therefore, our designed controller is still inadequate in heading error control. However, our method is more stable, and the relatively more important lateral error control effect is better than the other two controllers.



(a) (b)

Fig. 7: Evaluation of preset trajectory tracking in the simulator (a) Statistics for lateral errors,(b)Statistics for heading errors

Fig. 8: Planning environment

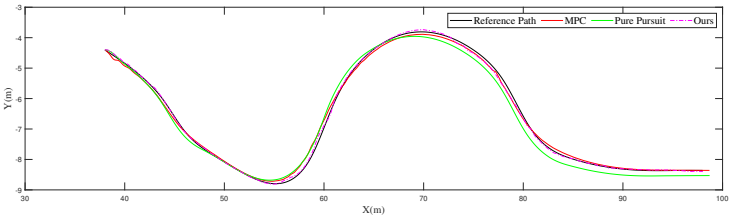
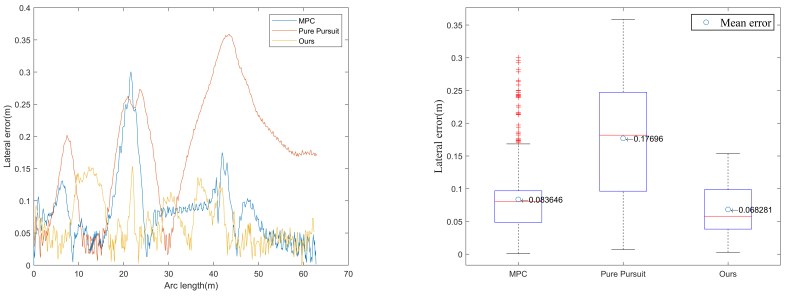


Fig. 9: Effect of planning trajectory tracking

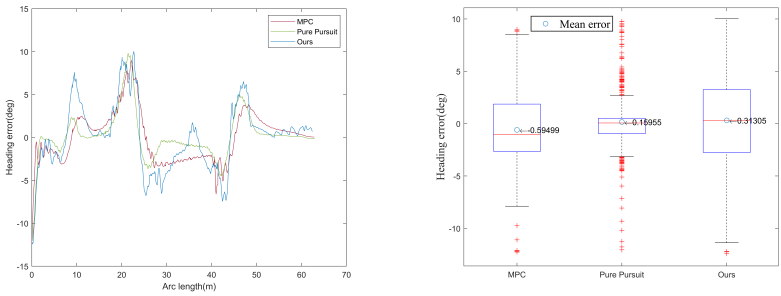
4.3.2 Planning trajectory tracking

In the task of planning trajectory tracking, as illustrated in Figure. 8, we created a simulation environment in Gazebo to model the lane change scenario for unmanned vehicles. The reference trajectory is generated by the front-end planner, and then we evaluate the tracking performance of three trajectory tracking algorithms as shown in Figure.9. The lateral error and heading error statistics during the tracking process are shown in Figure.10 and 11. As observed in the figure, the average value of the heading error of the proposed method is similar to the other two methods. However, the lateral distance statistics closely follow the desired path, indicating minimal changes in lateral error and more stable lateral tracking.



(a) (b)

Fig. 10: Statistics for lateral errors. (a) Variation of lateral error with arc length,(b)Box plot of lateral error



(a) (b)

Fig. 11: Statistics for heading errors (a) Variation of heading error with arc length,(b)Box plot of heading error

Table 5 presents the results of our performance comparison among different trajectory tracking approaches in terms of time. Notably, the Pure Pursuit method exhibits a shorter completion time and response time compared to the other two methods, owing to its mechanism of minimizing the lateral error via adjusting the front wheel rotation angle during tracking. However, this approach suffers from poor performance in terms of lateral errors. On the other hand, the completion time of the MPC method is similar to our proposed approach. However, its response time is considerably higher than the other two methods, with a maximum response time of up to 150.76 ms. Conversely, our proposed method exhibits a response time compared to the Pure Pursuit method but achieves superior tracking performance in lateral error. These findings indicate that our method is capable of providing rapid and accurate trajectory tracking performance in complex scenarios.

Table 5: Evalution time in the simulator

Method	Response Time(ms)			Travel Time(s)
	Max	Mean	Min	Mean
Ours	4.67	2.76	2.37	69.49
MPC	150.76	41.14	33.64	69.40
Pure pursuit	0.17	0.06	0.03	63.10

5 Conclusion

In this paper, a trajectory tracking controller based on the Dueling DDQN reinforcement learning method is proposed. This approach involves separating the state information encoding network from the unmanned vehicle policy

network and introducing a self-attention mechanism to achieve effective feature extraction and encoding of state information. To train our approach, we establish an interactive environment using the Gazebo simulation and design a multi-task intensive reward function for trajectory tracking.

The comparison experiments show that the reinforcement learning trajectory tracking controller proposed in this paper has an excellent performance in terms of response time and lateral error compared with the traditional pure pursuit trajectory tracking method and MPC based method. However, it does not improve much in heading error and is not suitable for scenarios with high requirements for heading error. In future research, discrete acceleration can be considered as the input to increase the controllable range of vehicle speed and enhance heading tracking performance.

Acknowledgments. This work was supported by Natural Science Foundation of Hebei Province (F2022203040, F2020203016), Natural Science Foundation of China (62073277, 62188101), Central Government Guided Local Science and Technology Development Fund Project (226Z0301G), Key R & D Program of Hebei Province (20311803D), Hebei Innovation Capability Improvement Plan Project (22567619H).

Declarations

- **Conflict of interest/Competing interests** Not applicable
- **Ethics approval** Not applicable
- **Consent to participate** Not applicable
- **Consent for publication** Not applicable
- **Availability of data and materials** Not available
- **Code** Not available
- **Authors'contributions**: **Weiming Liao:** Methodology, Software, Writing-Original draft preparation. **Guanglei Zhou:** Research ideas, Writing-Reviewing. **Zihao Chen:** Data curation,Investigation.

References

[1] B. Ravi Kiran, Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A.A., Yogamani, S., Pérez, P.: Deep reinforcement learning for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems (2021)

[2] Chen, J., Li, S.E., Tomizuka, M.: Interpretable End-to-End Urban Autonomous Driving With Latent Deep Reinforcement Learning. IEEE Transactions on Intelligent Transportation Systems **23**(6), 5068–5078 (2022). https://doi.org/10.1109/TITS.2020.3046646

[3] Huang, Z., Zhang, J., Tian, R., Zhang, Y.: End-to-End Autonomous Driving Decision Based on Deep Reinforcement Learning. In: 2019 5th

International Conference on Control, Automation and Robotics (ICCAR), pp. 658–662 (2019). https://doi.org/10.1109/ICCAR.2019.8813431

[4] Hilleli, B., El-Yaniv, R.: Toward deep reinforcement learning without a simulator: An autonomous steering example. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)

[5] Pivtoraiko, M., Knepper, R.A., Kelly, A.: Differentially constrained mobile robot motion planning in state lattices. Journal of Field Robotics **26**(3), 308–333 (2009). https://doi.org/10.1002/rob.20285

[6] Ammar, H.H., Azar, A.T.: Robust Path Tracking of Mobile Robot Using Fractional Order PID Controller. In: Hassanien, A.E., Azar, A.T., Gaber, T., Bhatnagar, R., F. Tolba, M. (eds.) The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2019). Advances in Intelligent Systems and Computing, pp. 370–381. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-14118-9_37

[7] Samuel, M., Hussein, M., Mohamad, M.B.: A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle. International Journal of Computer Applications **135**(1), 35–38 (2016). https://doi.org/10.5120/ijca2016908314

[8] Snider, J.M.: Automatic steering methods for autonomous automobile path tracking. Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08 (2009)

[9] Amer, N.H., Zamzuri, H., Hudha, K., Kadir, Z.A.: Modelling and Control Strategies in Path Tracking Control for Autonomous Ground Vehicles: A Review of State of the Art and Challenges. Journal of Intelligent & Robotic Systems **86**(2), 225–254 (2017). https://doi.org/10.1007/s10846-016-0442-0

[10] Tomatsu, T., Nonaka, K., Sekiguchi, K., Suzuki, K.: Model predictive trajectory tracking control for hydraulic excavator on digging operation. In: 2015 IEEE Conference on Control Applications (CCA), pp. 1136–1141 (2015). https://doi.org/10.1109/CCA.2015.7320765

[11] Chen, S.-p., Xiong, G.-m., Chen, H.-y., Negrut, D.: MPC-based path tracking with PID speed control for high-speed autonomous vehicles considering time-optimal travel. Journal of Central South University **27** (2020). https://doi.org/10.1007/s11771-020-4561-1

[12] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)

https://arxiv.org/abs/1604.07316

[13] Muller, U., Ben, J., Cosatto, E., Flepp, B., Cun, Y.: Off-road obstacle avoidance through end-to-end learning. Advances in neural information processing systems **18** (2005)

[14] Spielberg, N.A., Brown, M., Kapania, N.R., Kegelman, J.C., Gerdes, J.C.: Neural network vehicle models for high-performance automated driving. Science robotics **4**(28), 1975 (2019)

[15] Zhao, Y., Qi, X., Ma, Y., Li, Z., Malekian, R., Sotelo, M.A.: Path Following Optimization for an Underactuated USV Using Smoothly-Convergent Deep Reinforcement Learning. IEEE Transactions on Intelligent Transportation Systems **22**(10), 6208–6220 (2021). https://doi.org/10.1109/TITS.2020.2989352

[16] Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)

[17] Da Silva, F.L., Costa, A.H.R.: A survey on transfer learning for multiagent reinforcement learning systems. Journal of Artificial Intelligence Research **64**, 645–703 (2019)

[18] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous Control with Deep Reinforcement Learning. arXiv (2019). https://doi.org/10.48550/arXiv.1509.02971

[19] Konda, V., Tsitsiklis, J.: Actor-critic algorithms. Advances in neural information processing systems **12** (1999)

[20] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: International Conference on Machine Learning, pp. 387–395 (2014). Pmlr

[21] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. arXiv (2013). https://doi.org/10.48550/arXiv.1312.5602

[22] Fujimoto, S., Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning, pp. 1587–1596 (2018). PMLR

[23] He, R., Lv, H., Zhang, S., Zhang, D., Zhang, H.: Lane Following Method Based on Improved DDPG Algorithm. Sensors **21**(14), 4827 (2021). https://doi.org/10.3390/s21144827

[24] Liu, M., Zhao, F., Yin, J., Niu, J., Liu, Y.: Reinforcement-Tracking: An Effective Trajectory Tracking and Navigation Method for Autonomous Urban Driving. IEEE Transactions on Intelligent Transportation Systems **23**(7), 6991–7007 (2022). https://doi.org/10.1109/TITS.2021.3066366

[25] Gu, S., Holly, E., Lillicrap, T., Levine, S.: Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 3389–3396 (2017). https://doi.org/10.1109/ICRA.2017.7989385

[26] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An Introduction to Deep Reinforcement Learning, (2018)

[27] Bellman, R.: A Markovian decision process. Journal of mathematics and mechanics, 679–684 (1957)

[28] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., de Freitas, N.: Dueling Network Architectures for Deep Reinforcement Learning. arXiv (2016)

[29] Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30 (2016)

[30] Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015) https://arxiv.org/abs/1511.05952

[31] Zhao, G., Sun, X., Xu, J., Zhang, Z., Luo, L.: MUSE: Parallel Multi-Scale Attention for Sequence to Sequence Learning. arXiv (2019)

[32] Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), vol. 3, pp. 2149–21543 (2004). https://doi.org/10.1109/IROS.2004.1389727