Article | Received 1 February 2025; Revised 26 September 2025; Accepted 9 October 2025; Published 4 November 2025
https://doi.org/10.55092/rl20250009

Multi-task learning for underwater robot via progressive neural network

Xingwei Pan, Song Xiang, Shilong Niu, Zheng Fang, Jun Wang and Guangliang Li*

College of Electronic Engineering, Ocean University of China, Qingdao, China

* Correspondence author; E-mail: guangliangli@ouc.edu.cn.

Highlights:

- Proposed a Progressive Neural Network (PNN) framework for multi-task learning of remotely operated vehicles (ROVs).
- Introduced a three-column Progressive Neural Network (SPNN) that dynamically selects relevant knowledge to improve learning efficiency.
- Evaluated on the Gazebo platform, SPNN outperforms PNN, SAC, and PID in learning speed and generalization to new tasks.

Abstract: Remotely operated vehicles (ROVs) have been popularly applied in a range of complex underwater tasks. To minimize the human operator's workload and errors, deep reinforcement learning has been introduced to realize autonomous control of ROVs. However, current research on ROV control via deep reinforcement learning mostly focuses on single task learning. In this paper, we proposed and implemented progressive neural networks (PNNs) for ROV multi-task learning. In addition, to improve the robot's learning efficiency, we further proposed a three-column PNN (SPNN) that dynamically selects the transferring experience from previous path following and obstacle avoidance tasks based on the detected situation in the new path following with local path planning task. We evaluated our methods on Gazebo with the BlueROV2 simulator in two types of tasks: straight line path following with local path planning, sinusoidal curve path following with local path planning. Our results show that ROV trained via PNNs and SPNN can learn to complete path following with local path planning tasks faster than directly transferring policy across multi-tasks (SAC-Multi) and directly trained policy in a single task (SAC-Single) via deep reinforcement learning SAC method, and generalize better to new tasks than the SAC-Multi, SAC-Single and the traditional PID controller.

Keywords: deep reinforcement learning; remotely operated vehicle; path following; path planning; progressive neural networks

1. Introduction

The remotely operated vehicle (ROV) is playing a significant role in ocean research and exploration, and has been widely used for underwater tasks, such as underwater surveys, operations and infrastructure

inspection [1]. Precisely controlling ROV's movement is crucial to replace humans to complete these marine tasks, especially those dangerous ones. Although the ROV is usually remotely operated, manual intervention with ROV depends on the experience of operators and will increase the workload of the operator, which greatly decreases the work efficiency [2]. Path following and planning are two important techniques for ROV to perform underwater tasks, and usually need to be executed at the same time to complete complex marine missions. Therefore, it is necessary to improve the autonomy and intelligence of the ROV, especially the ability to perform multiple tasks, which can reduce an operator's workload and errors during operations, thus improving its working efficiency.

Lots of traditional control methods have been proposed and implemented for ROV control. For example, the traditional Proportional–Integral–Derivative (PID) controller has been widely used for underwwater robot control due to its simplicity, ease of implementation, and technological maturity [3,4]. Chen *et al.* designed and validated model-based PID controllers for ROV depth and Yaw–Pitch–Roll (YPR) attitude angles [5]. However, PID controllers often require frequent parameter tuning to achieve precise control in changing marine environments, and their performance is easily affected by sensor noise and ocean currents in complex oceanic conditions. Amundsen *et al.* [6] proposed a line-of-sight (LOS) method to guide the ROV to autonomously traverse an aquaculture net pen. Hosseini *et al.* [7] proposed a sliding mode approach for ROV control to deal with model uncertainty and disturbances in path following tasks. Hong *et al.* [8] proposed an adaptive sliding mode controller for ROV trajectory tracking, but the proposed method requires accurate parameter estimation. However, the LOS and sliding mode controller usually cannot adapt to marine environmental changes. To deal with disturbing forces and moments generated by currents, cables and disturbances, Huo et al. proposed a fuzzy logic system (FLS) to approximate the forces and moments with an adaptive speed controller for hover and path-following control of ROV [9], but requiring lots of expertise of the system and task. Yang *et al.* proposed a DNN + ENMPC strategy that effectively enhances ROV trajectory tracking accuracy under various disturbances [10]. However, in their method, data usually needs to be collected beforehand for pretraining and lacks online adaptability.

Deep reinforcement learning (deep RL) is applied to realize the autonomous control of ROVs due to its advantages of not requiring prior knowledge of the environment and being able to adapt to marine environmental changes. Deep reinforcement learning combining deep learning and RL can realize an end-to-end autonomous control of ROVs using raw high-dimensional input [11,12]. For example, Khan *et al.* [13] used the Proximal Policy Optimization (PPO) algorithm for the ROV to complete the path following task. Cadengue *et al.* [14] proposed to use an intelligent controller to realize the accurate depth control of ROV, using the Upper Confidence Bound algorithm [15] to compensate for external disturbances and unmodeled dynamics. Fule *et al.* [16] proposed a RL-based integral sliding mode controller for depth determination control of ROV. Fan *et al.* [17] proposed an innovative ROV path-following control method by designing and integrating a novel experience replay strategy within the twin-delayed deep deterministic policy gradient algorithm (TD3). However, the above deep RL methods generally focused on single task learning and may not generalize to other tasks. For ROV control, it usually requires performing multiple tasks at the same time, e.g., tracking a target while avoiding obstacles. In these complex missions, multiple RL learning models would be needed.

Multi-task reinforcement learning (MTRL) allows using only one agent to learn to accomplish multiple tasks, aiming to bootstrap learning in one task by simultaneously learning on other related tasks [18–20].

Recent research focused on novel sophisticated architectures to improve the performance of multi-task learning, which often results in larger models. Progressive neural networks (PNNs) [21] as one of the most outstanding MTRL method, have shown strong potential for forward transfer across tasks by dynamically expanding network structures without requiring explicit alignment between source and target domains. PNNs can learn multiple tasks using several column networks with each column of the network solving one single task so as to avoid catastrophic forgetting in new tasks. By preserving previously acquired knowledge and isolating learning for new tasks, PNNs offer a powerful mechanism to adapt to new and even different environments. PNNs have been shown to be successful in a curriculum task learning and enable a robot arm to obtain all the performance of policy trained in previous tasks almost immediately [19,22]. In addition, Xu *et al.* proposed the FARPPO algorithm using progressive networks for AUV depth-tracking control, which enhances adaptability and robustness of AUV policy in varying dynamics with a two-stage training mechanism [23].

To reduce an operator's workload and improve the working efficiency, this paper proposes and implements a Progressive Neural Network (PNN) framework tailored for multi-task learning in remotely operated vehicles (ROVs). In addition, to make better use of knowledge and experience from previous learned tasks, we further propose a three-column Progressive Neural Network (SPNN), which dynamically selects the most relevant prior task knowledge based on the context detected during new path-following and local planning tasks. By adaptively transferring the most suitable prior experience according to the current state, the SPNN is expected to improve learning efficiency and performance in complex underwater multi-task scenarios. We evaluate our method on Gazebo with BlueROV2 simulator in two types of tasks: straight line path following with local path planning, sinusoidal curve path following with local path planning. We compared to deep reinforcement learning SAC method in the two tasks (SAC-Multi) and single tasks (SAC-Single) as well as the traditional PID controller. Moreover, to test the generalization, we tested the final policies trained in the above two tasks in three new tasks: straight line path following with local planning with equally distributed obstacles and dense obstacles, and a sinusoidal curve path following with local path planning with dense obstacles. Our results show that the ROV trained via PNN can learn to complete both path following and path planning tasks faster than deep reinforcement learning in multi-tasks (SAC-Multi) and single task (SAC-Single). Furthermore, our method was shown to generalize better to new tasks than the one trained via SAC-Multi, SAC-Single, while the traditional PID controller cannot perform both path following and path planning task well at the same time and generalize to different tasks. The contributions of our work can be summarized as below:

- Propose and implement a Progressive Neural Network (PNN) framework tailored for multi-task learning with remotely operated vehicles (ROVs);
- To make better use of knowledge and experience from previous learned tasks, we propose a three-column Progressive Neural Network (SPNN), which dynamically selects the most relevant prior task knowledge based on the context detected in the new task;
- Evaluate the proposed methods using the BlueROV2 simulator on the Gazebo platform, and compare to deep reinforcement learning SAC method as well as the traditional PID controller.

2. Background

2.1. Mathematical model of ROV

Our simulations are performed with a ROV simulator—BlueROV2—an open sourced underwater robot platform from BlueRobotics [24]. The shape of BlueROV2 is a rectangular structure with dimensions of 457 mm Ã 338 mm Ã 254 mm. It weighs 10–11 kg, can be operated at a maximum depth of 100 m with a maximum speed of 3 kn for a maximum endurance of 4 hours. Figure 1 shows the distribution of six thrusters for BlueROV2. The thruster 1, 2, 3 and 4 are horizontally oriented and responsible for heading and steering, and the thruster 5 and 6 are vertically oriented and responsible for surfacing. BlueROV2 was equipped with sensors like sonar and depth sensor, which can be used to perform various tasks such as path following, path planning *etc.*

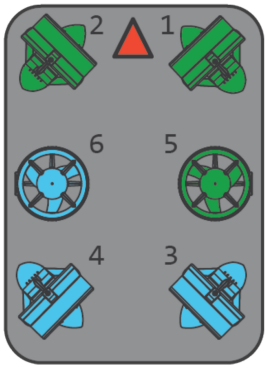


Figure 1. A distribution map of BlueROV2's thrusters, with thrusters 1 to 4 controlling horizontal movement, and thrusters 5 to 6 controlling vertical movement. Source: Reproduced from [24].

Figure 2 shows the world frame and the body frame of the ROV model separately [24]. We use the well-known compact $\eta - \nu$ Fossen notation [25], which collects the elements of the SNAME nomenclature in a vectorized format [26,27].

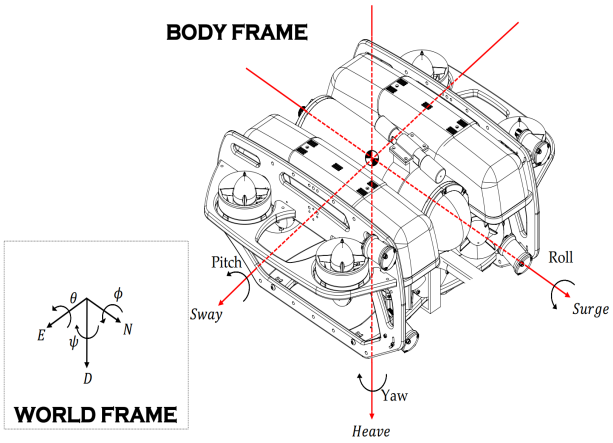


Figure 2. The body and world frome of BlueROV2. Source: Reproduced from [24].

The kinematic equations of BlueROV2 is described as:

$$\dot{\eta} = J(\eta)\nu, \tag{1}$$

where $\nu = [u, v, w, p, q, r]^T$ represents a vector consisting of body-fixed velocities (surge, sway, heave, roll,

pitch and yaw), u, v, w represent the linear velocities and p, q, r represent the angular velocities of the model. And $\eta = [x, y, z, \phi, \theta, \psi]^T$ is the combined vector that includes the positions and Euler angles relative to the n-frame. $J(\eta)$ is the coordinate transformation matrix, which can be expressed as:

$$J(\eta) = \begin{bmatrix} J_1(\eta) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & J_2(\eta) \end{bmatrix}, \tag{2}$$

where the matrix $J_1(\eta)$ characterizes the coordinate transformation of the ROV's linear velocity from the body-fixed frame to the inertial frame, while $J_2(\eta)$ characterizes the transformation of its angular velocity between the two frames.

The dynamics model of BlueROV2 is described as:

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \tau + \tau_{tet}, \tag{3}$$

where, the vector $\tau = [X, Y, Z, K, M, N]^T$ represents the applied forces and moments exerted on the vehicle, M represents the inertia coefficient matrix of the system, $C(\nu)$ denotes the Coriolis force coefficient matrix, $D(\nu)$ represents the viscous hydrodynamic coefficient matrix, $g(\eta)$ is the vector of restoring forces, and τ denotes the vector of control forces and moments generated by the ROV's control system, while τ_{tet} represents the vector of external disturbance forces and moments acting on the ROV during operation.

Specifically, the inertia matrix of the ROV is typically composed of the rigid-body inertia matrix M_{RB} and the added mass matrix M_A, and can be expressed as:

$$M = M_{RB} + M_A. \tag{4}$$

The Coriolis force coefficient matrix $C(\nu)$ is expressed as:

$$C(\nu) = C_{RB}(\nu) + C_A(\nu), \tag{5}$$

where $C_{RB}(\nu)$ represents the rigid-body Coriolis force coefficient matrix; $C_A(\nu)$ denotes the added mass Coriolis matrix. The hydrodynamic damping matrix $D(\nu)$ is expressed as:

$$D(\nu) = D + D_n(\nu), \tag{6}$$

where D represents the linear damping matrix of the ROV in six degrees of freedom, which is related to the linear and angular velocities; $D_n(\nu)$ denotes the nonlinear damping matrix of the ROV in six degrees of freedom, exhibiting a nonlinear dependence on the absolute values of the linear and angular velocities.

The ROV's motion is controlled through thrusters, typically driven by r individual thrusters. Therefore, the control vector τ can be related to the voltages applied to each thruster, expressed as follows:

$$\tau = TK(s)F(V), \tag{7}$$

where the matrix T denotes the thrust allocation matrix; the diagonal matrix $K(s)$ represents the unit DC gain transfer function matrix; V is the column vector of input voltages applied to the thrusters; $F(V)$ is the voltage-to-thrust conversion function that computes the generated force magnitude.

2.2. Markov decision process

A reinforcement learning agent usually can be modeled as a Markov decision process (MDP) [28]. A standard MDP can represented as $(S, A, P^a_{ss'}, R^a_{ss'}, \gamma)$. S is a set of states that the agent can visit, A denotes a set of actions that the agent can perform, $P^a_{ss'}$ denotes the transition probability from s to s' after choosing the action at time step t, and $R^a_{ss'}$ denotes the reward received by the agent after transitioning the state from s to s' via performing the selected action. γ is a discount factor within [0, 1], denoting the importance of the current and future rewards.

Specifically, at time step t, the agent interacts with the environment to obtain its state s. Then, it will selects an action a based on its current policy in current state s. After executing the action a, the state of the agent shifts from s to s', and it will receive a reward r. The agent will update its policy for choosing actions based on the state-action pair and the received reward, and move to a next round. The ultimate goal of policy updating is to learn an optimal control policy that maximizes the cumulative rewards.

The cumulative rewards are the sum of the rewards received by an agent via performing a series of actions in an MDP, where the discount factor is also taken into account, which can be expressed as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+(k+1)}. \tag{8}$$

The agent usually learns a state value function or an action value function. The state value function $V^\pi(s)$ is the expected cumulative rewards that an agent can obtain in the future, starting from state s, under policy π, which can be expressed as:

$$V^\pi(s) = \mathbb{E}_\pi[R_t | s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+(k+1)} | s_t = s \right]. \tag{9}$$

And the action value function $Q^\pi(s,a)$ measures the expected cumulative rewards an agent can obtain by taking an action a in a given state s under policy π, which can be expressed as:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t | s_i = s, a_t = a] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+(k+1)} | s_t = s, a_t = a \right]. \end{aligned} \tag{10}$$

The Bellman equation provides an iterative method for solving the optimal state value function and the optimal action value function. By iteratively updating the state-value function or action-value function, the optimal value function can be gradually approximated, thus obtaining the optimal policy by greedily selecting actions with the highest value. The Bellman optimality equation for the state value function is:

$$V^*(s) = max\mathbb{E}[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a]]. \tag{11}$$

And the Bellman optimality equation for action value function is:

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a} Q^*(s_{t+1}, a') | s_t = s, a_t = a \right]. \tag{12}$$

3. Our methodology

This section describes the principle of progressive neural networks (PNNs) for multi-task learning and our implementation of PNNs for ROV path following and planning.

3.1. Progressive Neural Networks for multi-task learning

Progressive Neural Networks (PNNs) represent an efficient approach to multi-task learning by introducing new network columns as the model transitions from single-task to multi-task settings. In this architecture, each new task is assigned a dedicated network column that leverages lateral connections to previously learned columns, enabling effective feature reuse and accelerating the learning process. This design enhances knowledge transfer across tasks while effectively avoiding catastrophic forgetting. By achieving a balance between parameter sharing and isolation, PNNs ensure that new tasks can benefit from prior knowledge without degrading the performance of previously learned tasks, thus maintaining learning stability. With excellent scalability, PNNs can flexibly accommodate an increasing number of tasks and dynamically adjust model complexity according to task difficulty. These features make PNNs particularly well-suited for learning in complex and dynamic environments, where robust multi-task adaptation is essential.

Progressive Neural Networks (PNNs) is a multi-column neural network, and each column has L layers with hidden activations $h_i^{(k)} \in \mathbb{R}^{n_i}$, and n_i is the number of units at layer $i \leq L$. PNNs can be used to continually learn to perform complex tasks or multiple tasks fast by reusing knowledge obtained in previous tasks. A two-column PNNs starts with a single column trained in a source task, which will be switched to a target task that is similar to or even different from the sournce one. The parameters $\Theta^{(1)}$ of the first column will be 'frozen' and the parameters of the second column are usually initialized with $\Theta^{(1)}$ or randomly. The activation $h_i^{(2)}$ of the second column layer will receive input from both $h_{i-1}^{(1)}$ and $h_{i-1}^{(2)}$. This can be extended to k-column:

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}\right), \tag{13}$$

where $W_i^{(k)} \in \mathbb{R}^{n_i \times n_{i-1}}$ is the weight matrix of layer i of column k, $U_i^{(k:j)} \in \mathbb{R}^{n_i \times n_j}$ are the lateral connections from layer i-1 of column j, to layer i of column k, and h_0 is the network input. f is an element-wise non-linearity: we use $f(x) = \max(0,x)$ for all intermediate layers. Figure 3 shows an example of three-column progressive neural networks.

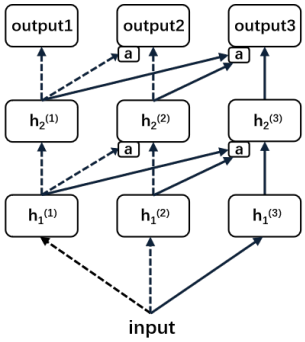


Figure 3. An example of three-column progressive neural network for multi-task learning (reproduced from [21]).

3.2. PNNs for ROV path following and planning

The progressive neural network (PNN) algorithm for ROV path following and planning employs a dual-column architecture, where the first column represents the pretrained path-following policy model, and the second column is the network to be trained in the new ROV path-following and planning task. With this structure, ROV trained with PNN is expected to make good use of knowledge and experience from pre-trained path following task to provide effective guidance for learning in the new path following with local planning task.

To be specific, at time step t in a path following task, ROV first obtained the state s_t by interacting with the marine environment, then used the initialized policy π_{ϕ_λ} to select and perform an action a_t. Then, ROV will receive a reward r_t from the defined reward function in Section 4.2.3 after performing the action a_t in the state s_t, and the environmental state of ROV will change to s_{t+1} at the time step $t+1$. The quaternion (s_t, a_t, s_{t+1}, r_t) will be used as a sample and stored in the replay buffer. At each step, 256 samples will be randomly selected from the replay buffer, and the MSEloss is computed and used to update the policy network π_{ϕ_λ}. The Soft Actor-Critic (SAC) algorithm [29,30] is employed for policy update. Policy update via SAC not only maximizes the cumulative rewards, but also maximizes its entropy:

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathscr{H}(\pi(\cdot|\mathbf{s}_t)) \right]. \tag{14}$$

The temperature parameter α was used to balance the entropy term and the reward in the objective function, thereby regulating the level of stochasticity in the optimal policy. The cycle of obtaining state, selecting and performing action, receiving reward and updating policy is repeated until an episode ends. Finally, the ROV learned an optimal policy π_{ϕ_λ} that performs the path following task well. After that, the policy π_{ϕ_λ} is used to initialize the first column of PNN and the second column of PNN is randomly initialized. We train ROV to obtain policy π_{ϕ_μ} represented by the second column of PNN to enable the ROV to perform obstacle avoidance while following a target path in the new task in a same way as the above process.

3.3. Selective progressive neural network for ROV path following and planning

Although the dual-column PNN can make use of the experience in the previous path following task to accelerate ROV learning to follow the target path in the new path following with obstacle avoidance task, it cannot benefit ROV learning to avoid obstacles in the new task. To make better use of knowledge and experience from previous path following and obstacle avoidance tasks, we further propose a three-column Progressive Neural Network (SPNN), which dynamically selects the most relevant prior task knowledge based on the context detected during new path-following with local planning task. By adaptively transferring the most suitable prior experience according to the current state, the SPNN is expected to improve learning efficiency and performance in complex underwater multi-task scenarios. The mechanism of SPNN is illustrated in Figure 4.

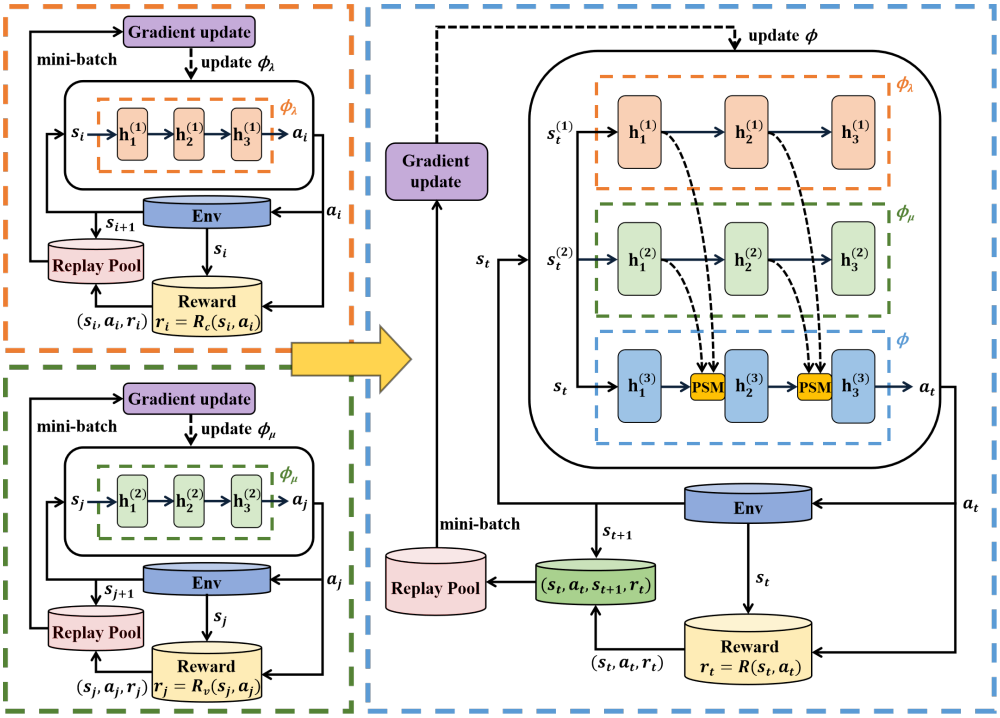


Figure 4. Illustration of the mechanism of our Selective Progressive Neural Networks (SPNN) method for ROV path following and planning. Note that $s_t{}^{(1)}$ and $s_t{}^{(2)}$ represent the state information for path following and obstacle avoidance, respectively.

In SPNN, as illustrated in Figure 4, the first column of PNNs is initialized with the policy π_{ϕ_λ} trained in the path following task which can follow a target path but cannot avoid obstacles, and the second column of PNNs is initialized with the policy π_{ϕ_μ} trained in the obstacle avoidance task which can avoid obstacles but cannot follow a target path. Meanwhile, the policy π_ϕ represented by the third column of SPNN for the path following with local path planning task is randomly initialized. When further training the ROV's policy in the new path following and path planning task, the first and second columns are 'frozen' and only the third column will be updated. In the SPNN structure, each layer of the third-column for policy π_ϕ receives inputs not only from the previous layer of its own column, but also from the corresponding layers of the first column for path-following and the second column for obstacle-avoidance. These inputs are combined through a parameter-sharing module that aggregates the information with a weighted sum before passing it to the next layer. In addition, during training the policy π_ϕ for the new task, the parameter-sharing module uses a selective mechanism which evaluates the ROV's current state and dynamically selects either π_{ϕ_λ} or π_{ϕ_μ} as the source of transferable experience. The unselected policy network does not contribute to the transfer at that time step. The specific rules governing the transfer of experience within the SPNN framework are as follows:

$$h_i^{(3)} = \begin{cases} \sigma * h_{i-1}^{(1)} + \tau * h_{i-1}^{(3)}, & d_{th} < d \\ \sigma * h_{i-1}^{(2)} + \tau * h_{i-1}^{(3)}, & d \le d_{th} \end{cases}, \tag{15}$$

where $h_i^{(3)}$ represents a layer of the ROV's policy π_ϕ, $h_{i-1}^{(3)}$ represents the previous layer of $h_i^{(3)}$ in the ROV's policy π_ϕ, and $h_{i-1}^{(1)}$ and $h_{i-1}^{(2)}$ are the layers of π_{ϕ_λ} in the first column for the path following task and π_{ϕ_μ} in the second column for the obstacle avoidance task, respectively, and the weight coefficients σ and τ are both set to be 0.5, d is the shortest distance between the ROV and the obstacle. d_{th} is the

threshold distance, which is set to be 12.5 m after a number of experimental trials. Specifically, when the ROV is farther than 12.5 m from the obstacles, the policy π_ϕ will be updated using parameters in the first column (*i.e.*, the policy π_{ϕ_λ} learned in the path following task) together with the new state; when the ROV is closer than 12.5 m to the obstacles, the policy π_ϕ will be updated using parameters in the second column (*i.e.*, the policy π_{ϕ_μ} learned in the obstacle avoidance task) together with the new state. We expect that through dynamic selection of network parameters from different tasks, the ROV can learn to perform the path following and planning task faster and better. The pseudo code for ROV path planning is shown in Algorithm 1.

Algorithm 1 SPNN for ROV path following and planning

Require:
The observation s_n of ROV
The action a_n of ROV
The next timestep observation s_{n+1} of ROV
The reward of environments $r_n \leftarrow \psi(s_n, a_n)$
Ensure:
Learned policy π_ϕ and Q function Q_θ of ROV
Initialize parameters ϕ, θ for π_ϕ, Q_θ
Initialize an empty replay pool D
//Control policy learning in path following via SAC
for $k = 0, 1, 2, ..., K$ **do**:
for $i = 0, 1, 2, ..., I$ **do**:
$(s_i, a_i, s_{i+1}) \leftarrow$ Path following environment
$r_i \leftarrow R(s_i, a_i)$;
Add (s_i, a_i, s_{i+1}, r_i) to D_1
end for
Update policy π_{ϕ_λ} and Q function Q_{θ_λ} via SAC
end for
//Control policy learning in obstacle avoidance via SAC
for $l = 0, 1, 2, ..., L$ **do**:
for $j = 0, 1, 2, ..., J$ **do**:
$(s_j, a_j, s_{j+1}) \leftarrow$ Obstacle avoidance environment
$r_j \leftarrow R(s_j, a_j)$;
Add (s_j, a_j, s_{j+1}, r_j) to D_2
end for
Update policy π_{ϕ_μ} and Q function Q_{θ_μ} via SAC
end for
//Control policy learning in path following and planning via SPNN
Transfer policy π_{ϕ_λ}, π_{ϕ_μ}, Q_{θ_λ}, Q_{θ_μ} to path planning task via three-column PNNs
Freeze the first-column and second-column of PNNs
for $e = 0, 1, 2, ..., E$ **do**:
for $t = 0, 1, 2, ..., T$ **do**:
$(s_t, a_t, s_{t+1}) \leftarrow$ Path following and planning environment
$r_t \leftarrow R(s_t, a_t)$
Add (s_t, a_t, s_{t+1}, r_t) to D
end for
Update policy π_ϕ of the third-column of PNNs and Q function Q_θ via SAC
end for

4. Simulations and results

In this section, we describe our simulation platform, experimental tasks, state and action space, reward functions, and results analysis.

4.1. Simulation platform

We used the Unmanned Underwater Vehicle (UUV) [31] simulator on the Gazebo platform to evaluate our method, as shown in Figure 5. Gazebo is an open source robot simulation platform for modeling and testing robot behavior and interactions with a variety of environments. It provides an accurate physics simulation engine and sensor simulation capabilities that are widely used in robotics research, development and education. The underwater environment on Gazebo can be configured to mimic the real environment, e.g., by placing obstacles, setting current speeds, *etc*. UUV Simulator is an open source simulation platform extended from Gazebo and dedicated to the simulation and study of unmanned underwater vehicles. It is based on the Robot Operating System (ROS) framework and provides a range of tools and models for simulating the motion, sensors, control and environment of underwater robots. The UUV Simulator provides a more realistic testing environment for the algorithm design with ROV.

As shown in Figure 5, we used the BlueROV2 model on UUV Simulator. It has six thrusters, four of which control ROV to move horizontally and the other two thrusters control ROV to move vertically. The ROV is equipped with a range of onboard sensors, such as sonar sensor, leak detection sensors, an Inertial Measurement Unit (IMU), a magnetometer, a pressure transmitter and a power sense module.

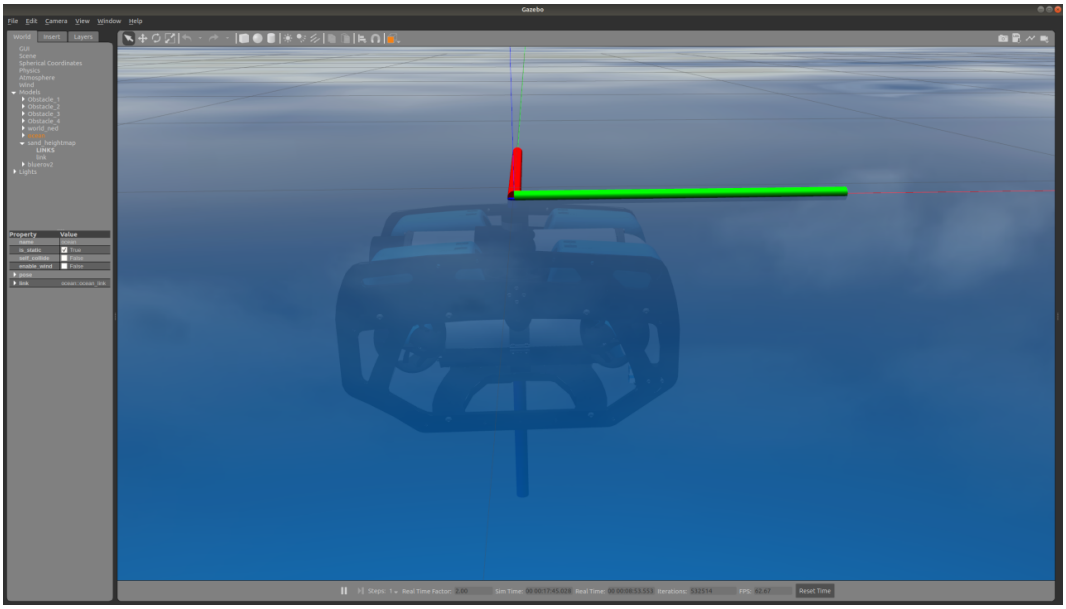


Figure 5. Screenshot of our our BlueROV2 simulator on Gazebo.

4.2. Simulation setup

We describe the setup for our simulations as below, including the tasks, state representation and action space, reward function design and simulation settings.

4.2.1. Experimental tasks

Two types of tasks are set to evaluate our methods: straight line path following with local path planning tasks, and sinusoidal curve following with local planning tasks.

In the straight line path following task, the ROV is required to follow a straight line of 200 meters in length, as shown in Figure 6. We set $\Omega = (x,y)|0 \leq x \leq 200, -30 \leq y \leq 30$ as the task area. The ROV starting state is at the point (0, 0), and the target point is at (200, 0). In the straight line path following with local path planning task, the target path is the same as in the straight line path following task, and simplified cylindrical obstacles are placed at the following coordinates: (66, −2.5), (80, 20), (118, −20), and (132, 2.5). These obstacles have a uniform radius of 1.5 meters. We use cylindrical obstacles to simplify the collision detection process while maintaining strong generality, making them suitable for most cases involving non-compact obstacles. When the ROV does not detect an obstacle while traveling, it continues following the straight line to complete the path tracking, and when it detects an obstacle in front of it, the ROV needs to avoid the obstacle and return to the set path to complete the task.

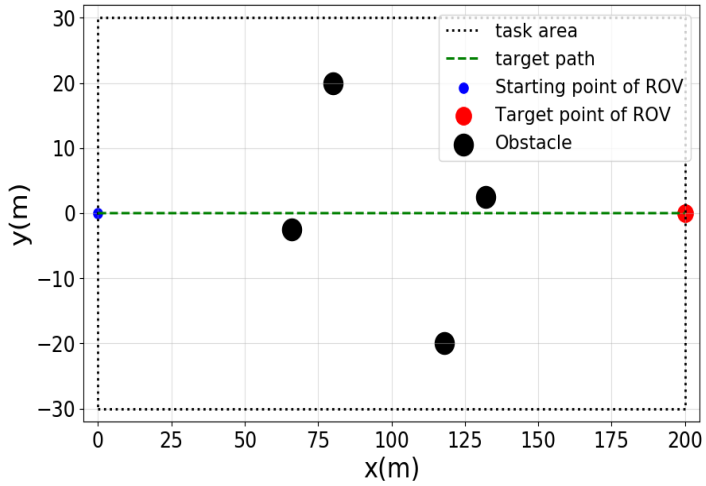


Figure 6. An illustration of the straight line path following with local path planning task in the underwater environment.

In the sinusoidal curve following task, the ROV is required to follow a sinusoidal curve with a length of 120 meters, as shown in Figure 7. We set $\Omega = (x,y)|0 \leq x \leq 120, -30 \leq y \leq 30$ as the task area. The ROV's starting state is at the point (0, 0), and the target point is at (120, 0). In the sinusoidal curve following with local path planning task, the target path is the same as in the sinusoidal curve following task. In addition, we set up the obstacles at the following locations: (30, 17.5), (90, −12.5). These obstacles have a uniform radius of 1.5 meters. The task is similar to the straight line path following with local path planning task: the ROV carries out a path following task along a sinusoidal curve, avoids obstacles along the way and then continue following the path to complete the task.

ROV is trained to complete the two tasks within one episode. If it meets the following conditions, ROV fails to finish the tasks and initiate a new episode:

(1) ROV is more than $|d_L|$ metres from the target path, with d_L = 30 for straight line following with local path planning tasks, and d_L = 15 for sinusoidal curve following with local path planning tasks;
(2) The heading angle deviation of the ROV is too large, *i.e.*, $|a_c| \geq \frac{\pi}{2}$.

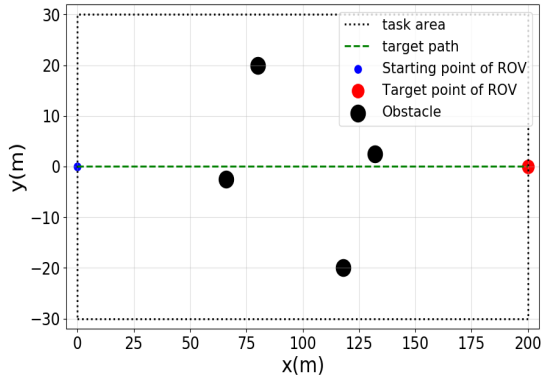


Figure 7. An illustration of the sinusoidal curve following with local path planning task in the underwater environment.

4.2.2. State and action space

In both kinds of path following with local path planning tasks, the ROV used a sonar sensor to detect obstacles, and the span of the detection angle is S. We divide the detection span of angle equally into q parts, so that each part of the sonar sensor's detection angle is S/q. Each part of the ROV's detection range specifies the shortest distance from an obstacle as d_i. The state for the ROV is represented as $o = [a_c, g, d_1, d_2, ..., d_q]$, as shown in Figures 8 and 9. a_c is the heading angle deviation, which is calculated as:

$$a_c = c - c_d, \tag{16}$$

where c represents the current heading angle and c_d represents the expected heading angle. g is the vertical distance from the target path. The ROV used line of sight (LOS) [32] to correct the course during path following in both types of tasks. In both path following tasks without obstacles, the state is represented as $[a_c, g]$.

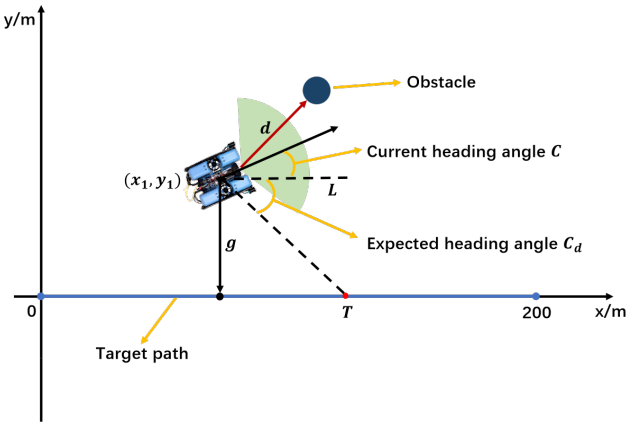


Figure 8. An illustration of the state representation for the ROV in the straight line following with local path planning task. (x_1, y_1) is the current position of ROV, d is the distance from the obstacle, L is the frontal distance, g is the vertical distance, T is the next target from the target path.

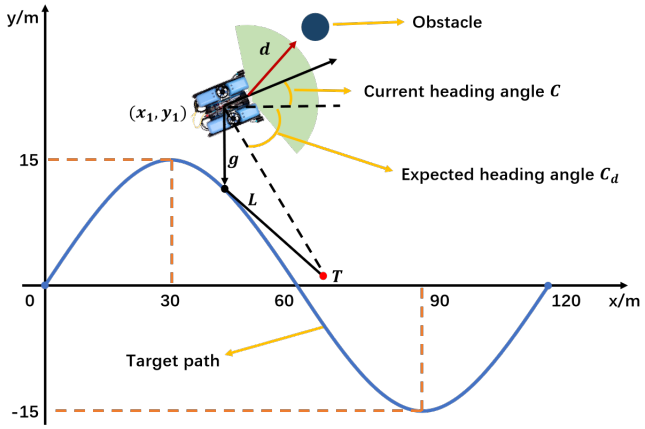


Figure 9. An illustration of the state representation for the ROV in the sinusoidal curve following with local path planning task. (x_1, y_1) is the current position of ROV, d is the distance from the obstacle, L is the frontal distance, g is the vertical distance, T is the next target from the target path.

In all above tasks, ROV is controlled by setting the speed of the thrusters. BlueROV2 has six thrusters, four of which (T1, T2, T3, T4) control the horizontal movements and two of which (T5, T6) control the vertical movements. As shown in Table 1, the action space is set to be discrete, with a total of five actions: turning right (two), turning left (two), and going straight. In our experiments, since the ROV only needs to complete a 2D task because its equipped sonar can only provide two-dimensional sensory information, we set the speed of T5 and T6 to be 0.

Table 1. Actions available to ROV in the 2D path following with local path planning tasks (The unit of T is rev/min).

Action	T1	T2	T3	T4	T5	T6
Turn left 1	0	6000	0	−6000	0	0
Turn left 2	0	8000	0	−8000	0	0
Go straight	8000	8000	−8000	−8000	0	0
Turn right 1	6000	0	−6000	0	0	0
Turn right 2	8000	0	−8000	0	0	0

4.2.3. Reward function

In our simulations, since the ROV needs to perform both path following and path planning at the same time, we designed reward functions composed of two parts:

$$r = r_c + r_v, \tag{17}$$

where r_c represents the reward for path tracking and r_v denotes reward for path planning. r_c is computed according to the heading angle deviation a_c and the distance from the target trajectory g. r_c for the straight line following with local path planning tasks is as below:

$$r_c = -\frac{|a_c|}{3.14} + \left|1 - \frac{g}{d_L}\right|, \tag{18}$$

and r_c for the sinusoidal curve following with local path planning tasks is as below:

$$r_c = \begin{cases} -|a_c| * 0.175 + 2^{2-\frac{|g|}{d_L}} * 0.325 & g \leq 1.8 \\ -|a_c| * 0.0875 + 2^{2-\frac{|g|}{d_L}} * 0.1625 & else, \end{cases} \tag{19}$$

with d_L = 30 for straight line path following with local path planning tasks and d_L = 15 for sinusoidal curve path following with local path planning tasks. r_v is calculated according to the distance from the obstacle in both types of tasks:

$$r_v = \begin{cases} -2, & 5 < d \leq 8 \\ -4, & 2 < d \leq 5 \end{cases}. \tag{20}$$

In addition, in both types of tasks, the ROV will receive another reward when certain conditions are satisfied:

$$r = \begin{cases} -200, & d_L \leq |g| \\ -200, & \frac{\pi}{2} \leq |a_c| \\ -200, & d \leq 2 \end{cases}. \tag{21}$$

4.2.4. Simulation settings

In the simulations, all parameters are experimentally optimized in multi-task learning scenarios. As shown in Table 2, for policy training in each task, there are one actor network and two critic networks: a main Q-network and a target Q-network. All networks are fully connected neural networks (FCNs): FCN = [255, 255, 2] for the actor network, and FCN = [255, 255, 1] for the critic network. The ReLU activation was used for the hidden layers, with no activation function in the final output layer. The learning rate of the policy network is set to be 0.0003, and the learning rates of the main Q-network and the target Q-network are set to be 0.0003 and 0.005, respectively. As shown in Table 3, γ is set to be 0.99, the hyper-parameter α is initially set to be 0.2, the size of the experience pool is set to 1,000,000. At the end of each episode, we will randomly take 256 samples from the experience pool for 10 times to update the policy network.

Table 2. Network parameters.

Network	Structure (FCN)	Learning Rate	Hidden Layer Activation Function	Output Layer Activation Function
Actor Network	[255, 255, 2]	0.0003	ReLU	None
Main Q-Network	[255, 255, 1]	0.0003	ReLU	None
Target Q-Network	[255, 255, 1]	0.005	ReLU	None

Table 3. Training hyperparameters.

Parameter	Value
Discount Factor (γ)	0.99
Initial Temperature Parameter (α)	0.2
Experience Pool Size	1,000,000
Number of Samples per Episode	10
Number of Samples per Sampling	256

4.3. Results and analysis

In this section, we presents and analyzes results of the ROV trained using our PNN agent, SPNN agent, in comparison with the SAC-Multi agent and SAC-Single agent. The SAC-Single agent is directly trained in the straight line and sinusoidal curve path following with local path planning tasks, respectively. In contrast, both our PNN agent and the SAC-Multi agent were first trained in a path following task via SAC. However, the SAC-Multi agent was directly transferred to the corresponding path following with local path planning task, while our PNN agent was transferred to the path following with local path planning task via a two-column progressive neural network. In addition, our SPNN agent was first trained in a path following task and an obstacle avoidance task, respectively, and then transferred to the path following with local path planning task via a three-column progressive neural network.

4.3.1. Straight line path following with local path planning

Figure 10a shows the learning curves of our PNN, SPNN, SAC-Single, SAC-Multi agent training in the straight line path following with local path planning task in terms of cumulative episode rewards according to the reward function defined in Section 4.2.3. Figure 10a shows that, at the beginning, the learning speed of the SAC-Multi agent and our PNN and SPNN agents is faster than that of the SAC-Single agent, since they already learned a good policy to follow the straight line in the previous path following task. However, the learning speed of the SAC-Multi agent decreased dramatically afterwards, which might be because of the new situations with obstacles in the path following with local path planning task, and kept a similar speed to that of the SAC-Single agent. In contrast, our PNN agent learned still much faster than both SAC-Multi and SAC-Single agents, and converged to a higher performance faster than both of them. Moreover, our SPNN agent learned a best performance at a faster speed than other agents, since it can dynamically select the transferring experience from previous path following and obstacle avoidance tasks based on the detected situation in the new path following with local path planning task.

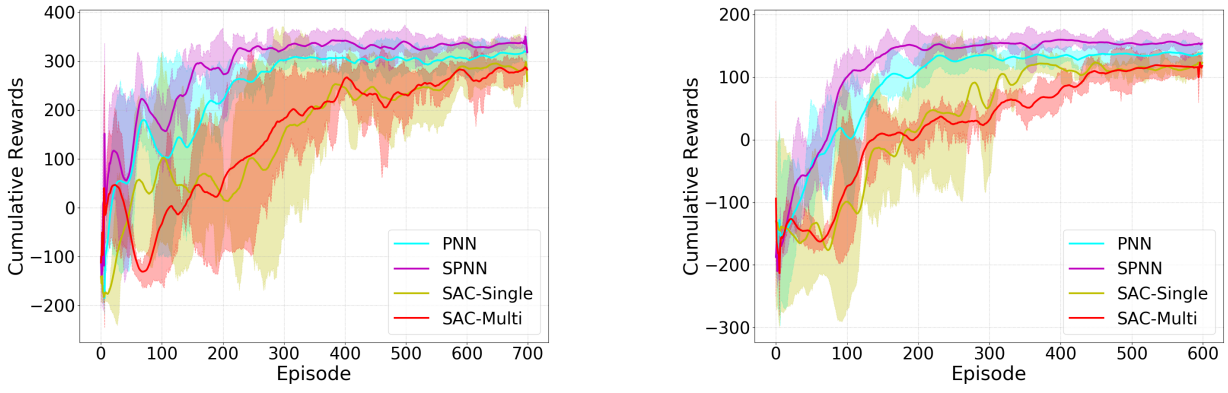


Figure 10. Cumulative rewards received by the PNN, SPNN, SAC-Multi and SAC-Single agents in the ROV straight line path following with **(a)** local path planning task and sinusoidal curve path following with **(b)** local path planning task (averaged over data collected in three trials). Note that the shaded area is the 0.95 confidence interval and the bold line is the mean performance.

Figure 11 shows the trajectories of the ROV trained via PNN, SPNN, SAC-Single and SAC-Multi during the training process in the straight line path following with local path planning task. A PID controller that performs well in the straight line path following task was tested in the straight line path

following with local path planning task for comparison. In addition, an adaptive PID controller with the same parameters as the PID controller but tuned for the straight line path following with local path planning task is used as baseline and shown in Figure 11e. As shown in Figure 11, at the beginning, all four agents cannot finish the task at all. After 80 episodes' training, the SAC-Multi and SAC-Single agents still cannot finish the task, while the PNN agent and SPNN agent can complete the task but do not follow the line well. At Episode 150, the SAC-Multi and SAC-Single agents still cannot finish the task, while the PNN agent and SPNN agent can avoid the obstacles and follow the line with fluctuations. At Episode 230, our SPNN agent is already able to perform the path following with local path planning task well. At Episode 250, the SAC-Single agent can finish the task without following the target line, the SAC-Multi agent can follow the target line and avoid obstacles with larger fluctuations than our PNN agent. After 300 Episodes' training, our PNN agent can approach the performance of our SPNN agent at Episode 230, while both the SAC-Multi and SAC-Single agents can complete the task with much larger deviations from the target line. In addition, the PID controller that performed well in the straight line path following task cannot finish the straight line path following with local path planning task at all, which indicates that a traditional PID controller cannot perform both path following and path planning task well at the same time.

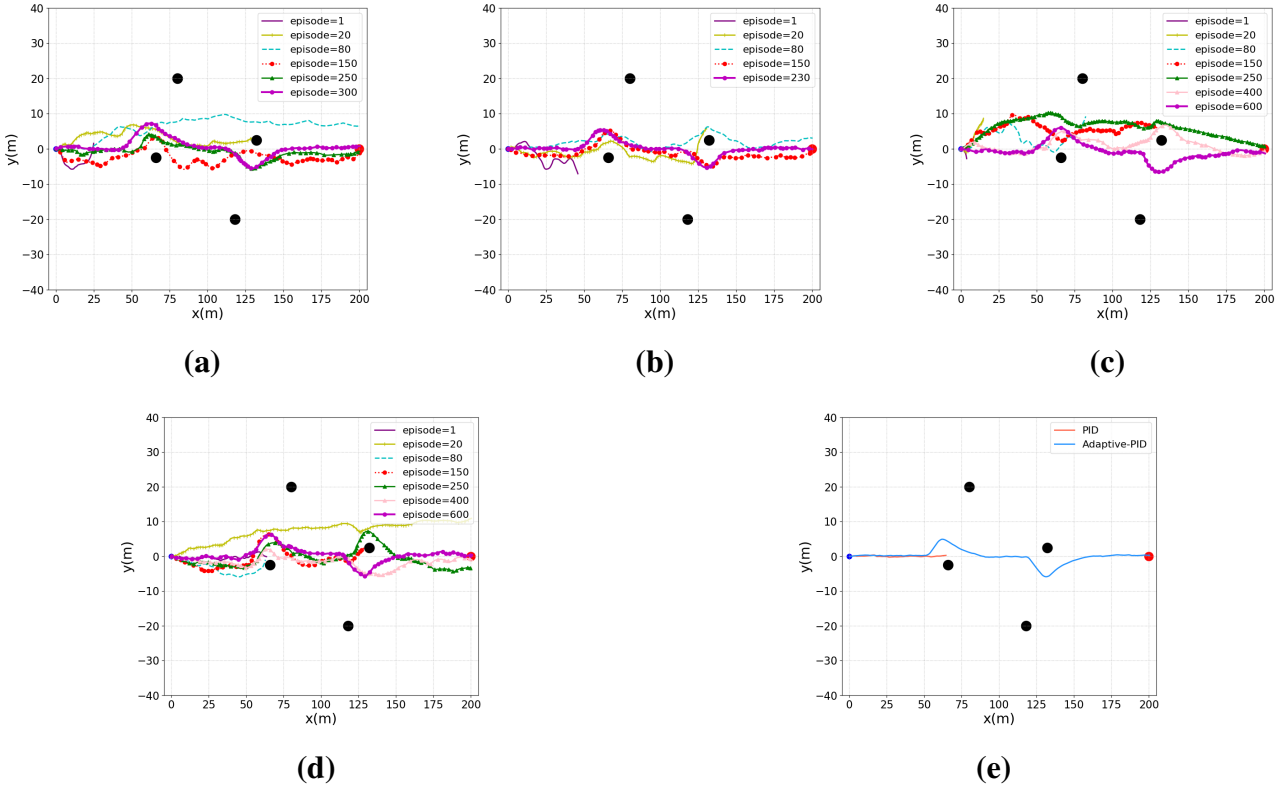
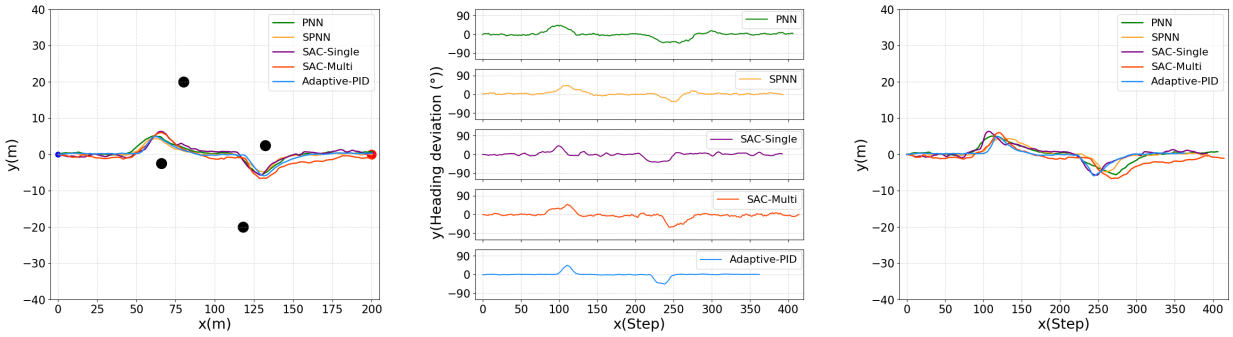


Figure 11. Trajectories of ROV trained with our method **(a)** PNN, **(b)** SPNN and **(c)** SAC-Single, **(d)** SAC-Multi during the training process in the straight line path following with local path planning task, in comparison with **(e)** a PID controller that performing well in the straight line path following task and an adaptive PID controller with the same parameters as the PID controller but tuned for the straight line path following with local path planning task.

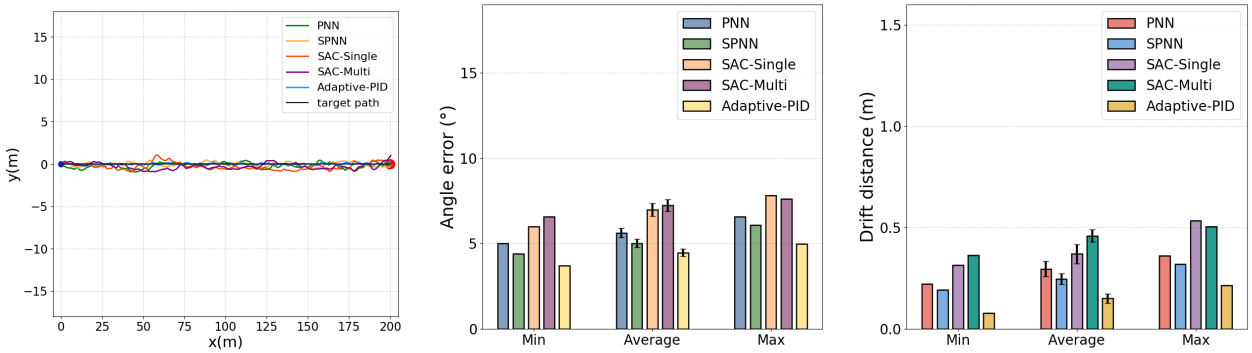
We also compared the final performance of the four agents in terms of the tested trajectories, heading deviation and distance from target route, in the straight line path following with local path planning, and the path following task without obstacles, using final policies trained via PNN, SPNN, SAC-Single

and SAC-Multi in the straight line path following with local path planning task, in comparison with the adaptive PID controller that performed the straight line path following with local path planning task well, as shown in Figures 12 and 13, respectively. From Figures 12 and 13 we can see that, while both SAC-Multi and SAC-Single agents could finish both the straight line path following with local path planning task and the path following task with fluctuations, our SPNN and PNN agents can complete both tasks with smaller heading deviation and distance from target route, reaching a performance close to the adaptive PID controller in the two tasks.



(a) **(b)** **(c)**

Figure 12. Tested **(a)** trajectories, **(b)** heading deviation, and **(c)** distance from the target route of the ROV in the straight line path following with local path planning task, using final policies trained via PNN, SPNN, SAC-Single and SAC-Multi in the same task, in comparison to the adaptive PID controller performing the straight line path following with local path planning task well.



(a) **(b)** **(c)**

Figure 13. Tested **(a)** trajectories, **(b)** heading deviation and **(c)** distance from the target route of the ROV in the straight line path following task, using final policies trained via PNN, SPNN, SAC-Single and SAC-Multi in the straight line path following with local path planning task, in comparison to the adaptive PID controller performing the straight line path following with local path planning task well.

4.3.2. Sinusoidal curve path following with local path planning

Figure 10b shows the learning curves of our PNN, SPNN, SAC-Single, SAC-Multi agent training in the sinusoidal curve path following with local path planning task measured by cumulative rewards received within one episode based on the defined reward function in Section 4.2.3. It can be seen from Figure 10b that, the learning speed of the SAC-Multi agent is similar to that of the SAC-Single agent and even a bit slower during the training process, while our PNN agent and SPNN agent learnmuch faster than

both SAC-Multi and SAC-Single agents, and convergeto a higher performance faster than both of them. Moreover, our SPNN agent achieves the best performance at a faster speed than other agents.

Figure 14 shows the trajectories of the ROV trained via PNN, SPNN, SAC-Single and SAC-Multi in the sinusoidal curve path following with local path planning task. A PID controller that performed well in the sinusoidal curve path following task was tested in the sinusoidal curve path following with local path planning task for comparison. In addition, an adaptive PID controller with the same parameters as the PID controller but tuned for the sinusoidal curve path following with local path planning task is used as baseline and shown in Figure 14e. As shown in Figure 14, none of the four agents can finish the task at the beginning. After 80 episodes' training, only the SPNN agent can complete the task. After 140 episodes' training, the SAC-Multi and SAC-Single agents still cannot finish the task, while the PNN agent can complete the task with deviations from the target route and the SPNN agent can complete the task better than the PNN agent. At Episode 180, the SPNN agent achieves optimal performance. At Episode 220, the SAC-Multi and SAC-Single agents can finish the task with fluctuations, and PNN agent achieves optimal performance. After about 400 episodes' training, the SAC-Single agent and the SAC-Multi agent can achieve a performance similar to that of the PNN agent at Episode 220, which is almost as good as SPNN agent and the adaptive PID controller. In addition, the PID controller that performed well in the sinusoidal curve path following task cannot finish the sinusoidal curve path following with local path planning task at all, which indicates that a traditional PID controller is not able to perform both path following and path planning task well at the same time.

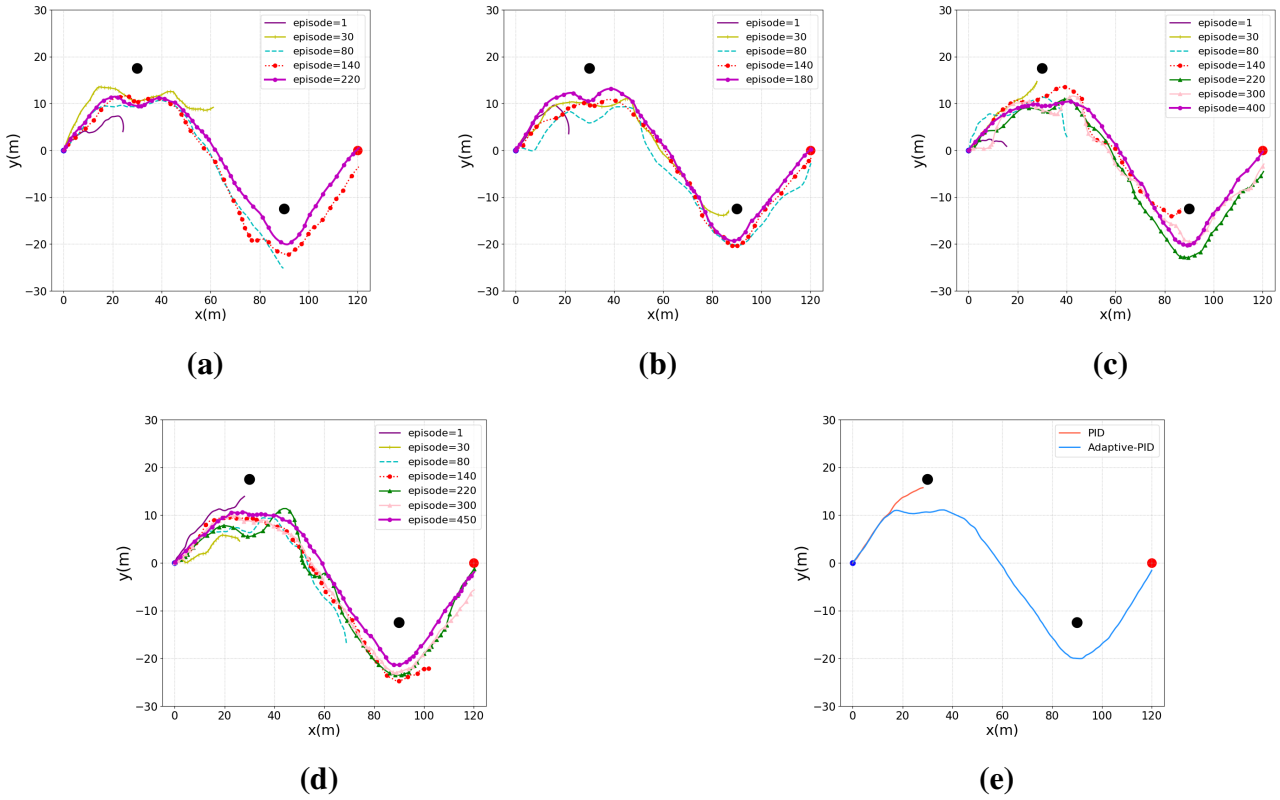
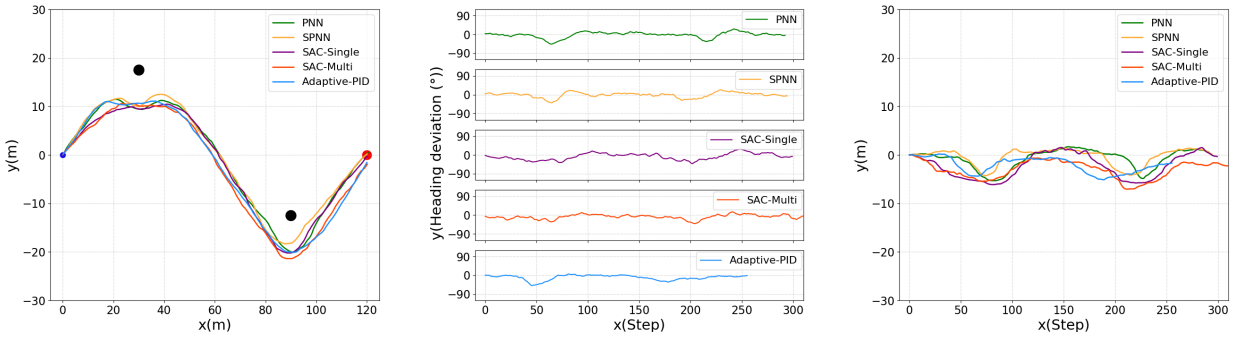


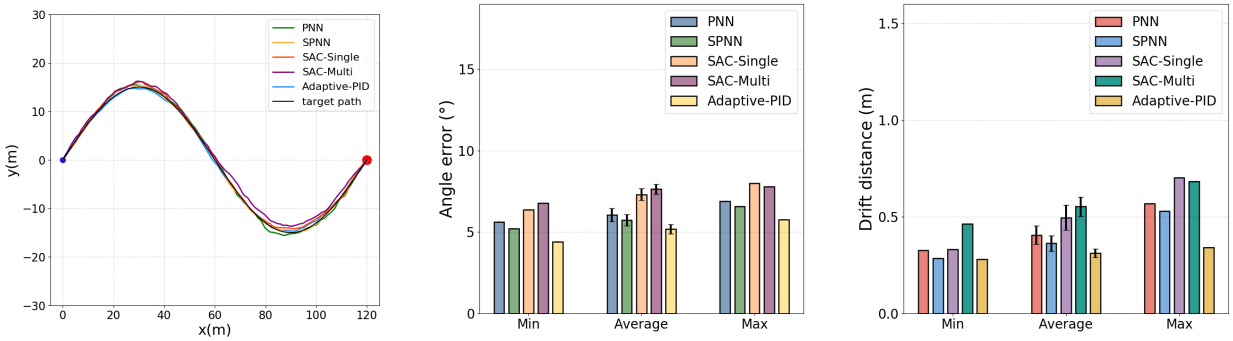
Figure 14. Trajectories of ROV trained via our method **(a)** PNN, **(b)** SPNN, **(c)** SAC-Single, **(d)** SAC-Multi during the training process in the sinusoidal curve path following with local path planning task, in comparison to **(e)** a PID controller performing well in the sinusoidal curve path following task and an adaptive PID controller with the same parameters but tuned for the sinusoidal curve path following with local path planning task.

The final performance of the four agents is also compared in terms of the tested trajectories, distance from target path and heading deviation in both the sinusoidal curve following with local path planning task and path following tasks using final policies trained via PNN, SPNN, SAC-Single and SAC-Multi in the sinusoidal curve following with local path planning task, in comparison to the adaptive PID controller that performed the sinusoidal curve path following with local path planning task well, as shown in Figures 15 and 16, respectively. From Figures 15 and 16 we can see that, although both SAC-Multi and SAC-Single agents could finish both the sinusoidal curve following with local planning task and path following task, our PNN and SPNN agents can complete both tasks with smaller heading deviation and distance from target route, reaching a performance close to the adaptive PID controller in the two tasks.



(a) **(b)** **(c)**

Figure 15. Tested **(a)** trajectories, **(b)** heading deviation, and **(c)** distance from the target route of the ROV in the sinusoidal curve path following with local path planning task, using final policies trained via PNN, SPNN, SAC-Single, and SAC-Multi, in the same task, in comparison to the adaptive PID controller performing the sinusoidal curve following with local path planning task well.



(a) **(b)** **(c)**

Figure 16. Tested **(a)** trajectories, **(b)** heading deviation and **(c)** distance from the target path in the sinusoidal curve following task, using final policies trained via PNN, SPNN, SAC-Single and SAC-Multi in the sinusoidal curve following with local path planning task, in comparison to the adaptive PID controller performing the sinusoidal curve following with local path planning task well.

4.3.3. Generalization to new tasks

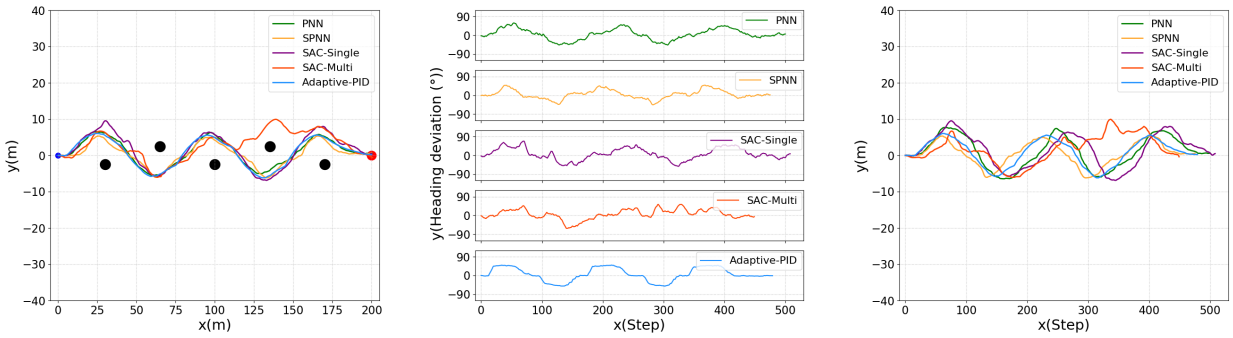
We also evaluated the generalization of our PNN agent and SPNN agent by testing and comparing final policies of the ROV trained via our PNN, SPNN, SAC-Multi and SAC-Single in three new path following with local path planning tasks. Specifically, we saved the final policies of the ROV trained via the four

agents in the original straight line path following with local path planning task and tested them in two new straight line following with local path planning tasks: one with uniformly distributed obstacles along the target route, one with dense obstacles along the target route. The adaptive PID controller that performed the original straight line path following with local path planning task well was also tested in the two new tasks for comparison. Moreover, we saved the final policies of the ROV trained via the four agents in the original sinusoidal curve path following with local path planning task and tested them in a new sinusoidal curve following with local path planning task with dense obstacles along the target route. The adaptive PID controller performing the original sinusoidal path following with local path planning task well was also tested in the new task for comparison.

Figures 17, 18 and 19 show the tested trajectories, distance from the target path and heading deviation of the four agents in the three new tasks. From Figures 17, 18 and 19 we can see that, while all PNN, SPNN, SAC-Single and SAC-Multi agents can finish the three new tasks, our SPNN agent can perform the new tasks with smaller fluctuations, distance from the target path and heading deviation compared to the PNN, SAC-Single and SAC-Multi agents. In contrast, the adaptive PID controller that performed well in the original tasks can complete the new straight line planning task with uniformly distributed obstacles with larger fluctuations than the SPNN agent, and cannot finish the new straight line planning task with dense obstacles and new sinusoidal curve planning task at all.

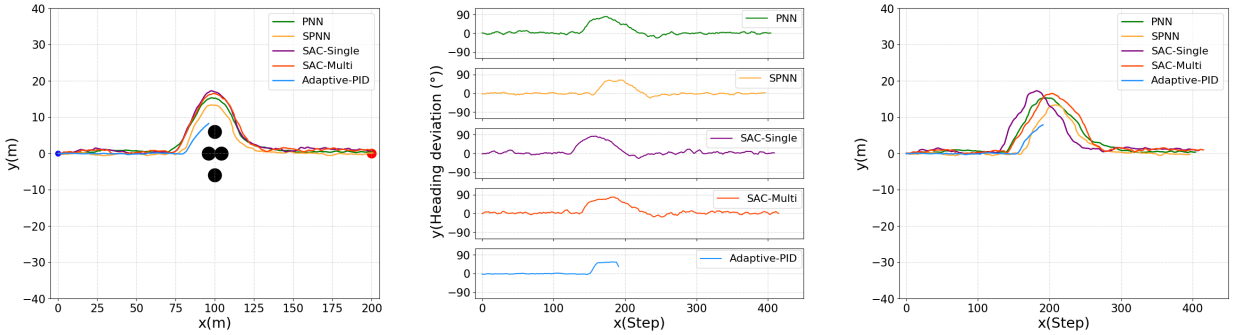
Our results in Figure 20 show that the final control policies of all four agents and adaptive PID controller can complete the new straight line path following with local path planning task with uniformly distributed obstacles along the target route with high success rate (SPNN: 100%, PNN: 98%, Adaptive PID: 96%, SAC-Single: 88%, SAC-Multi: 90%), which might be because this new task is similar to the original straight line planning task. In the new straight line planning task with dense obstacles along the target route which is quite different from the original task, the success rate of completing the task by both SAC-Single and SAC-Multi agents decreased to 80% and 66%, and the adaptive PID controller has only 11% success rate, while our PNN and SPNN agent still kept 92% and 98% success rate, respectively. In the new sinusoidal curve planning task with dense obstacles that is different from the original sinusoidal curve planning task, the success rate of completing the task by both SAC-Multi and SAC-Single agents decreased to about 80% and the adaptive PID controller has only 31% success rate, while our PNN and SPNN agents still kept 94% and 98% success rate, respectively.

In summary, our results indicate that the ROV trained via our PNN and SPNN can complete both path following and path planning tasks well, and generalize better to new tasks than the one trained via SAC-Multi, SAC-Single. Moreover, our SPNN agent can obtain aneven better performance than the PNN agent faster, since it can dynamically select the transferring experience from previous path following and obstacle avoidance tasks based on the detected situation in the new path following with local path planning task.



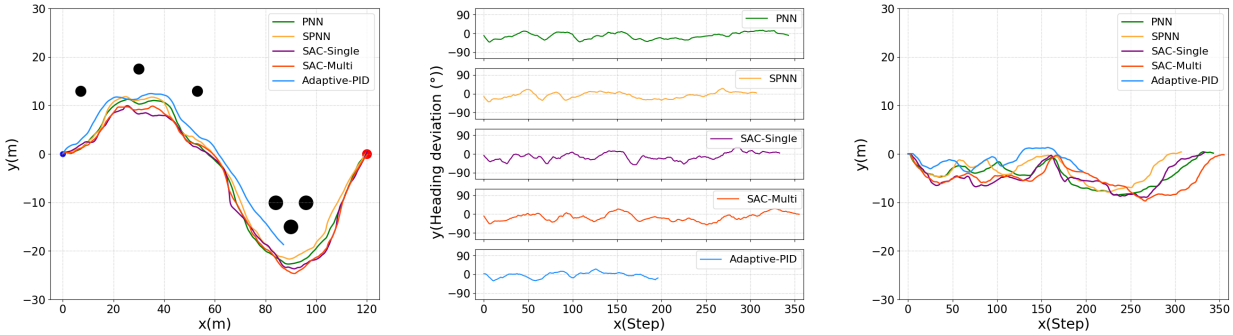
(a) **(b)** **(c)**

Figure 17. Tested **(a)** trajectories, **(b)** heading deviation and **(c)** distance from the target route of the ROV in a new straight line path following with local path planning task with uniform distributed obstacles along the target route, using final policies trained via PNN, SPNN, SAC-Single and SAC-Multi, together with the adaptive PID controller performing the original straight line path following with locak path planning task well.



(a) **(b)** **(c)**

Figure 18. Tested **(a)** trajectories, **(b)** heading deviation, and **(c)** distance from the target route of the ROV in a new straight line path following with local path planning task with dense obstacles along the target route, using final policies trained via PNN, SPNN, SAC-Single, and SAC-Multi, together with the adaptive PID controller performed the original straight line path following with local path planning task well.



(a) **(b)** **(c)**

Figure 19. Tested **(a)** trajectories, **(b)** heading deviation, and **(c)** distance from the target route of the ROV in a new sinusoidal curve following with local planning task with more and dense obstacles along the target route, using final policies trained via PNN, SPNN, SAC-Single, and SAC-Multi, together with the adaptive PID controller performing the original sinusoidal curve following with local path planning task well.

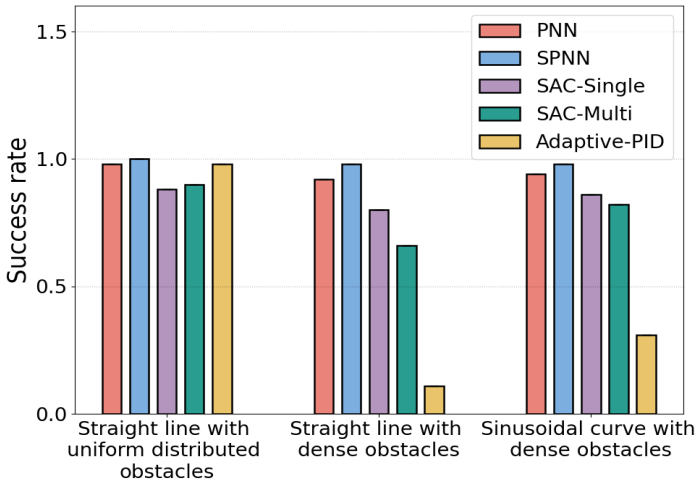


Figure 20. Success rate of completing three new planning tasks.

5. Conclusion

In this paper, we proposed and implemented progressive neural networks (PNN) for ROV multi-task learning. In addition, we further proposed a three-column PNN (SPNN) that dynamically selects the transferring experience from previous path following and obstacle avoidance tasks based on the detected situation in the new path following with local path planning task. We evaluated our methods with the BlueROV2 simulator on Gazebo in two types of tasks: straight line path following with local path planning, sinusoidal curve path following with local path planning, and compared to deep reinforcement learning SAC method in the two tasks (SAC-Multi) and single tasks (SAC-Single) as well as the traditional PID controller. Moreover, to evaluate the generalization to new situations, we tested the final policies trained in the above two tasks in three new and complex tasks: straight line path following with local planning task with equally distributed obstacles and dense obstacles, and a sinusoidal curve path following with local path planning task with dense obstacles. Our results indicate that the ROV trained via PNN and SPNN can learn to complete both path following with local path planning tasks faster than deep reinforcement learning in multi-tasks (SAC-Multi) and single task (SAC-Single), and generalize better to new situations than the one trained via SAC-Multi, SAC-Single and the traditional PID controller.

Although our proposed PNN and SPNN methods are capable of learning to accomplish multiple tasks and adapt to changes in the marine environments, the tracking accuracy of our methods can be further improved compared to the finely tuned adaptive PID controller for specific and fixed task. Moreover, in our SPNN method, we use a fixed threshold as the switching criterion for transferring between different task policies which imposes limitations on its generalization to other tasks. Future work plan to allow SPNN to dynamically adjust the weights for balancing experience transfer from previous different tasks.

In addition, our simulations are currently limited to two 2D tasks because of sonar sensor and are more applicable to ROVs without tether-induced drag effects or to untethered underwater vehicles (AUVs). In future work, we aim to investigate motion planning methods that account for tether dynamics, as the tether is a critical constraint for ROVs and may introduce drag and entanglement risks that significantly impact control performance. Moreover, we plan to extend our approach to a wider range of environments and three-dimensional tasks, including obstacle avoidance under varying ocean currents and path-following with local planning in the presence of dense, irregular, or even dynamic obstacles. We will

also evaluate the our method's robustness to diverse and challenging situations, such as cluttered seabed terrains and environments with variable buoyancy. Comprehensive metrics evaluating ROV performance such as energy efficiency and response time are also necessary for practical usage of our method. Finally, we would like to extend our methods to continuous control of ROV, transfer and validate on the physical ROV system in the marine environment with sim-to-real transfer methods [19,33–35] and handling additional real-world factors like water currents and sensor noise.

Data availability statement

The data or datasets that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments

This work was supported by Young Taishan Scholars Program (under Grant tsqn202408072) and in part by Qingdao Natural Science Foundation (under Grant No. 23-2-1-153-zyyd-jch).

Authors' contribution

Xingwei Pan: methodology, formal analysis, data curation, writing original draft. Song Xiang: formal analysis, data curation, visualization, writing—review & editing, validation. Shilong Niu: investigation, resources, writing—review & editing. Zheng Fang: software, formal analysis, writing—review & editing, visualization. Jun Wang: resources, review & editing. Guangliang Li: supervision, review & editing, project administration, funding acquisition. All authors have read and agreed to the published version of the manuscript.

Conflicts of interests

The authors declare no conflict of interest.

References

[1] Das P. The future of underwater robotics: trends and technologies in ROV design and application. *Int. J. Multidiscip. Res.* 2023, 5(4):1–7.

[2] Di Lillo PA, Simetti E, De Palma D, Cataldi E, Indiveri G, *et al.* Advanced ROV autonomy for efficient remote control in the DexROV project. *Mar. Technol. Soc. J.* 2016, 50(4):67–80.

[3] Anderlini E, Parker GG, Thomas G. Control of a ROV carrying an object. *Ocean Eng.* 2018, 165:307–318.

[4] Dong J, Duan X. A robust control via a fuzzy system with PID for the ROV. *Sensors* 2023, 23(2):821.

[5] Chen Y, Sun Z, Chen Y. Research on motion control system of underwater remotely operated vehicles. In *Sixth Conference on Frontiers in Optical Imaging and Technology: Applications of Imaging Technologies*, Nanjing, China, October 22–24, 2023, pp. 305–312.

[6] Amundsen HB, Caharija W, Pettersen KY. Autonomous ROV inspections of aquaculture net pens using DVL. *IEEE J. Oceanic Eng.* 2022, 47(1):1–19.

[7] Hosseini M, Seyedtabaii S. Robust ROV path following considering disturbance and measurement error using data fusion. *Appl. Ocean Res.* 2016, 54:67–72.

[8] Hoang NQ, Kreuzer E. A robust adaptive sliding mode controller for remotely operated vehicles. *Tech. Mech.* 2008, 28(3–4):185–193.

[9] Huo X, Ge T, Wang X. Horizontal path-following control for deep-sea work-class ROVs based on a fuzzy logic system. *Ships Offshore Struct.* 2018, 13(6):637–648.

[10] Yang G, Liu W, Li L, Xu J, Guo L, *et al.* Optimized trajectory tracking for ROVs using DNN + ENMPC strategy. *J. Mar. Sci. Eng.* 2024, 12(10):1827.

[11] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, *et al.* Human-level control through deep reinforcement learning. *Nature* 2015, 518(7540):529–533.

[12] Cheng C, Sha Q, He B, Li G. Path planning and obstacle avoidance for AUV: a review. *Ocean Eng.* 2021, 235:109355.

[13] Khan A. Deep reinforcement learning based tracking behavior for underwater vehicles. Master's Thesis, NTNU, 2018.

[14] Cadengue LS, Lima GS, Bessa WM. Intelligent depth control of underwater robots using artificial neural networks and reinforcement learning. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, Natal, Brazil, November 9–13, 2020, pp. 1–5.

[15] Hao B, Abbasi Yadkori Y, Wen Z, Cheng G. Bootstrapping upper confidence bound. In *Advances in Neural Information Processing Systems 32*, Vancouver, Canada, December 8–14, 2019.

[16] Wang F, Qu Q, Yuan B, Sun L, Li Y, *et al.* Optimal sliding mode control of ROV fixed depth attitude based on reinforcement learning. In *2021 IEEE 11th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, Jiaxing, China, July 27–31, 2021, pp. 79–84.

[17] Fan Y, Dong H, Zhao X, Denissenko P. Path-following control of unmanned underwater vehicle based on an improved TD3 deep reinforcement learning. *IEEE Trans. Control Syst. Technol.* 2024, 32(5):1904–1919.

[18] Tomov MS, Schulz E, Gershman SJ. Multi-task reinforcement learning in humans. *Nat. Hum. Behav.* 2021, 5(6):764–773.

[19] Ju H, Juan R, Gomez R, Nakamura K, Li G. Transferring policy of deep reinforcement learning from simulation to reality for robotics. *Nat. Mach. Intell.* 2022, 4(12):1077–1087.

[20] McLean R, Chatzaroulas E, Terry J, Woungang I, Farsad N, *et al.* Multi-task reinforcement learning enables parameter scaling. *arXiv* 2025, arXiv:2503.05126.

[21] Rusu AA, Rabinowitz NC, Desjardins G, Soyer H, Kirkpatrick J, *et al.* Progressive neural networks. *arXiv* 2016, arXiv:1606.04671.

[22] Rusu AA, Večerík M, Rothörl T, Heess N, Pascanu R, *et al.* Sim-to-real robot learning from pixels with progressive nets. In *Proceedings of the 1st Annual Conference on Robot Learning*, California, USA, November 13–15, 2017, pp. 262–270.

[23] Xu C, Fang T, Xu D, Yang S, Zhang Q, *et al.* A fast adaptive AUV control policy based on progressive networks with context information. *J. Mar. Sci. Eng.* 2024, 12(12):2159.

[24] von Benzon M, Sørensen FF, Uth E, Jouffroy J, Liniger J, *et al.* An open-source benchmark simulator: control of a BlueROV2 underwater robot. *J. Mar. Sci. Eng.* 2022, 10(12):1898.

[25] Fossen TI. Nonlinear modelling and control of underwater vehicles. Master's Thesis, Universitetet i Trondheim (Norway), 1991.

[26] Engineers S. Nomenclature for treating the motion of a submerged body through a fluid. 1950. Available: https://www.scribd.com/document/57744791/Nomenclature-for-Treating-the-Motion-of-a-Submerged-Body-Through-a-Fluid (accessed on 15 December 2024).

[27] Gertler M, Hagen GR. Standard equations of motion for submarine simulation. 1967. Available: https://apps.dtic.mil/sti/tr/pdf/AD0653861.pdf (accessed on 15 December 2024).

[28] Sutton RS, Barto AG. Reinforcement learning: an introduction, 2nd ed. Massachusetts: MIT press, 2018.

[29] Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, July 10–15, 2018, pp. 1861–1870.

[30] Haarnoja T, Zhou A, Hartikainen K, Tucker G, Ha S, *et al.* Soft actor-critic algorithms and applications. *arXiv* 2018, arXiv:1812.05905.

[31] Manhães MMM, Scherer SA, Voss M, Douat LR, Rauschenbach T. UUV simulator: a gazebo-based package for underwater intervention and multi-robot simulation. In *OCEANS 2016 MTS/IEEE Monterey*, Monterey, USA, September 19–23, 2016, pp. 1–8.

[32] Han P, Liu Z, Zhou Z, Tang H, Ban I, *et al.* Path tracking control algorithm based on LOS method for surface self-propulsion vessel. *Appl. Sci. Technol.* 2018, 45(3):66–70.

[33] Juan R, Huang J, Gomez R, Nakamura K, Sha Q, *et al.* Shaping progressive net of reinforcement learning for policy transfer with human evaluative feedback. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, September 27–October 1, 2021, pp. 1281–1288.

[34] Juan R, Ju H, Huang J, Gomez R, Nakamura K, *et al.* Sim-to-real policy and reward transfer with adaptive forward dynamics model. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, May 29–June 2, 2023, pp. 7212–7218.

[35] Meng W, Ju H, Ai T, Gomez R, Nichols E, *et al.* Transferring meta-policy from simulation to reality via progressive neural network. *IEEE Rob. Autom. Lett.* 2024, 9(4):3696–3703.