



Article

Adaptive Position Control of Pneumatic Continuum Manipulator Based on MAML Meta-Reinforcement Learning

Lina Hao *, Qiang Cheng , Hongshuai Liu  and Ying Zhang

Department of Mechanical Engineering and Automation, Northeastern University, Shenyang 110819, China; qcheng1022@163.com (Q.C.); liuhongshuai_1995@163.com (H.L.); zhangying@me.neu.edu.cn (Y.Z.)

* Correspondence: haolina@me.neu.edu.cn

Abstract: Reinforcement learning algorithms usually focus on a specific task, which often performs well only in the training environment. When the task changes, its performance drops significantly, with the algorithm lacking the ability to adapt to new environments and tasks. For the position control of a pneumatic continuum manipulator (PCM), there is a high degree of similarity between tasks, and the training speed of new tasks can be accelerated by utilizing the training experience from other tasks. To increase the adaptability of control policies to new tasks, this paper proposes an adaptive position control algorithm of the PCM based on the Model-Agnostic Meta-Learning (MAML) meta-reinforcement learning algorithm. The MAML meta-reinforcement learning algorithm is used to train the control strategy of PCM, and the information and experience collected during the training process across multiple tasks are used to quickly learn the control policy for new tasks, improving the ability of PCM to quickly adapt to new tasks. The experimental results demonstrate that after training with the MAML meta-reinforcement learning algorithm, the PCM can significantly reduce the training time when faced with new tasks and obtain the control policies suitable for these new tasks.

Keywords: pneumatic continuum manipulator; position control; MAML meta-reinforcement learning



Citation: Hao, L.; Cheng, Q.; Liu, H.; Zhang, Y. Adaptive Position Control of Pneumatic Continuum Manipulator Based on MAML Meta-Reinforcement Learning. *Appl. Sci.* **2024**, *14*, 10821. <https://doi.org/10.3390/app142310821>

Academic Editor: Grigorios Beligiannis

Received: 3 October 2024

Revised: 6 November 2024

Accepted: 21 November 2024

Published: 22 November 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The continuum manipulator is a bionic, slender super-redundant manipulator that provides superior reach and dexterity capabilities in cluttered environments. However, because of its high nonlinearity, the precise control of a continuum manipulator still remains a significant challenge.

Reinforcement learning is a model-free control method and has been used for the control of continuum manipulators. You [1] and Sreeshankar [2] used the Q-Learning algorithm and DQN algorithm to train the control policy for a planar continuum manipulator, achieving the position control of the manipulator. Sreeshankar [3] and Li [4] applied the Deep Deterministic Policy Gradient (DDPG) algorithm to the control of a continuum manipulator, achieving end-effector path tracking control. Centurelli [5] used the Trust Region Policy Optimization (TRPO) algorithm to learn the controller policy of the continuum manipulator, achieving position control of the continuum manipulator.

Although reinforcement learning algorithms have achieved great success in PCM control, they ignore the similarities between tasks, making them unable to effectively utilize training experiences from other tasks. Consequently, when faced with new tasks, the learning process often has to start from scratch, consuming substantial time and resources [6]. Therefore, how to use previous experience to improve the adaptability of agents to new tasks has become an important research direction. In order to enable the agent to have the ability to learn and adapt to new tasks quickly, researchers have proposed various solutions. Policy Distillation [7] extracts experience from multiple policies to form a more generalized comprehensive policy. This method can improve the adaptability of the strategy through

self-distillation or cross-task distillation. Multi-task learning [8] improves the generalization ability of an agent by simultaneously learning on multiple related tasks. However, the success of the above methods usually depends on the correlation between tasks. If the differences between tasks are too great, the effect of experience sharing may not be obvious, and it may even interfere with the learning process.

Some scholars have proposed the concept of meta-reinforcement learning, combining meta-learning algorithms with reinforcement learning algorithms based on the ability of meta-learning to rapidly learn with only a small number of samples [9]. Meta-reinforcement learning trains on multiple tasks, allowing agents to quickly adapt to new tasks with a small number of learning samples. Unlike traditional reinforcement learning, which optimizes control strategies for a single task, meta-reinforcement learning uses multiple tasks as its training samples. By training on multiple tasks, the agent acquires the ability to learn quickly. When faced with a new task, it is possible to rapidly learn the policy that adapts to the new task with only a small number of training samples by leveraging prior knowledge.

Typical meta-reinforcement learning algorithms include the memory context-based RL^2 algorithm [10], the SNAIL algorithm [11], and the CMRL algorithm [12]. These methods are easily combined with other learning algorithms and have a variety of network structures. However, complex learning tasks may require more complex models to process data and tasks, as well as more computational resources and training time. There are also task-inference-based algorithms, such as PEARL [13], MetaCURE [14], and MQL [15]. However, such methods may ignore some specific information related to the task during the task inference process, and their reliance on specific environmental conditions makes them difficult to generalize across different tasks [16]. Additionally, there are gradient descent-based algorithms, such as MAML meta-reinforcement learning [17]. As a novel meta-learning framework, MAML has spawned numerous improvements and extensions, including FO-MAML, ES-MAML [18], Reptile [19], MAESN [20], and Taming MAML [21]. The advantage of these algorithms lies in their model-agnostic nature, allowing them to be applied in various reinforcement learning scenarios and offering good generalization performance.

At the same time, there are also some works that applied meta-reinforcement learning algorithms to the control of manipulators. Li [22] combined the MQL algorithm with DDPG to propose a sample-efficient and generalizable method for manipulator skill learning. This method uses information such as the state of the end-effector and target pose as input to learn a policy that controls the manipulator's end-effector to reach the desired pose. Hao [23] combined the MAML meta-learning algorithm with the proximal policy optimization algorithm to design the M-PPO meta-reinforcement learning algorithm for learning the task of turning on a light with a manipulator. Schoettler [24] studied meta-reinforcement learning for industrial assembly tasks, using the PEARL algorithm to learn the latent representations of a task, enabling the rapid generation of policy for the current task based on its representation. Peng [25] integrated the Particle Swarm Optimization (PSO) algorithm with MAML, proposing the PSO-MAML algorithm. This algorithm was applied to the learning of grasping tasks, enabling the manipulator to quickly adapt to new tasks. Wang [26] proposed an offline policy meta-reinforcement learning algorithm SMRL-TO and applied it to the policy training of pushing tasks, door opening tasks, and assembly tasks of the manipulator, successfully achieving rapid adaptation of the manipulator to new tasks.

Meta-reinforcement learning has demonstrated strong generalization capabilities in the application of manipulators, but it has not yet been applied to the control of continuum manipulators. Because of the model-agnostic nature of the MAML algorithm, it can be applied in different reinforcement learning scenarios, and it exhibits good generalization performance. This paper employs the MAML meta-reinforcement learning algorithm to train the control policy of pneumatic continuum manipulators, improving the ability of pneumatic continuum manipulators to rapidly adapt to new tasks.

2. MAML Meta-Reinforcement Learning

The core objective of MAML meta-reinforcement learning is to train a set of initial policy parameters such that the policy initialized with these parameters can adapt to a new task rapidly by fine-tuning the parameters with minimal interaction dates.

The training process of MAML meta-reinforcement learning involves an inner reinforcement learning loop and an outer reinforcement learning loop.

Within the inner loop, the agent randomly samples K different training tasks \mathcal{T}_i from the task distribution $p(\mathcal{T})$ and interacts with the environment for each task \mathcal{T}_i to collect data \mathcal{D}_i . Let π_θ represent the agent's policy, where θ represents the policy parameters. The interaction process generates an action a_t based on the observed current state s_t according to the policy. The environment moves to the next state s_{t+1} and generates a reward r_t after the agent executes the action. The expected reward for each task \mathcal{T}_i can be expressed as

$$\mathcal{L}_{\mathcal{T}_i}(f_\theta) = E_{x_t, a_t \sim f_\theta, \mathcal{T}_i} \left[\sum_{t=1}^H r_t(x_t, a_t) \right] \quad (1)$$

The policy parameters are updated by maximizing the expected reward in the inner loop, and the update formula can be expressed as

$$\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \quad (2)$$

Within the outer loop, the parameters of the policy are updated according to the total expected rewards collected from each task. Within the inner loop, the agent interacts with the environment for each task \mathcal{T}_i to collect data \mathcal{D}'_i using the new policy $\pi_{\theta'_i}$, which was trained in the inner loop. Sum up the expected return $\mathcal{L}_{\mathcal{T}'_i}(f_{\theta'_i})$ for each task to obtain $\sum_{i=1}^K \mathcal{L}_{\mathcal{T}'_i}(f_{\theta'_i})$. The policy parameters are updated by maximizing the sum of the expected returns $\sum_{i=1}^K \mathcal{L}_{\mathcal{T}'_i}(f_{\theta'_i})$, and the update formula can be expressed as

$$\theta = \theta - \beta \nabla_\theta \sum_{i=1}^K \mathcal{L}_{\mathcal{T}'_i}(f_{\theta'_i}) \quad (3)$$

The overall training procedure of the MAML meta-reinforcement learning algorithm is illustrated in Algorithm 1.

Algorithm 1: MAML meta-reinforcement learning

Require: Task distribution

Require: Learning rates

1: Randomly initialize policy parameters

2: **while not done do:**

3: Sample a batch of tasks

4: **if all \mathcal{T}_i do:**

5: Interact with the environment using policy f_θ to collect data \mathcal{D}_i

6: Calculate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ based on \mathcal{D}_i and $\mathcal{L}_{\mathcal{T}_i}$

7: Update $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$

8: Interact with the environment using policy $f_{\theta'_i}$ to collect data \mathcal{D}'_i

9: **end for**

10: Update $\theta = \theta - \beta \nabla_\theta \sum_{i=1}^K \mathcal{L}_{\mathcal{T}'_i}(f_{\theta'_i})$

11: **end while**

2.1. Inner Reinforcement Learning Loop

The inner loop is updated by REINFORCE. The REINFORCE algorithm follows a straightforward idea of performing gradient ascent on the parameters of the policy θ to gradually improve the performance of the policy π_θ . The learning objective of REINFORCE is to maximize the expected reward, and the update formula can be expressed as

$$G_t = \sum_{k=t+1}^T \gamma^{k-t-1} r_k \tag{4}$$

where γ is the discount factor.

The policy parameters θ are updated by maximizing the expected reward using the gradient descent method. The update formula is as follows:

$$\theta_{t+1} = \theta_t - \eta G_t \ln \pi(a_t | s_t, \theta_t) \tag{5}$$

But, REINFORCE has been observed to suffer a large variance when estimating the gradient. To alleviate the large variance problem, a baseline $b(s_t)$ can be subtracted from the expected reward G_t , where $b(s_t)$ is a function only depending on s_t . One approach to selecting the baseline is to use the expected return starting from state s_t under the current policy as the baseline; that is, $b(s_t) = E[r_t + r_{t+1} + \dots + r_{T-1}]$. With the baseline $b(s_t)$, the update formula of policy parameters can be represented as

$$\theta_{t+1} = \theta_t - \eta (G_t - b(s_t)) \nabla_{\theta_t} \ln \pi(a_t | s_t, \theta_t) \tag{6}$$

2.2. Outer Reinforcement Learning Loop

The outer loop uses trust-region policy optimization (TRPO) as the optimizer. The goal of TRPO is to find an updated policy $\pi_{\theta'}$ that improves the current policy π_θ , that is, find better policy parameters θ' based on the current parameter θ to make $J(\theta') \geq J(\theta)$. $J(\theta)$ is the expected reward from the start state under the current policy π_θ , which can be denoted as

$$\begin{aligned} J(\theta) &= \mathbb{E}_{s_0} [V^{\pi_\theta}(s_0)] \\ &= \mathbb{E}_{\pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t V^{\pi_\theta}(s_t) - \sum_{t=1}^{\infty} \gamma^t V^{\pi_\theta}(s_t) \right] \\ &= -\mathbb{E}_{\pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t (\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)) \right] \end{aligned} \tag{7}$$

Define the advantage function $A^{\pi_\theta}(s_t, a_t)$ as

$$A^{\pi_\theta}(s_t, a_t) = r(s_t, a_t) + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t) \tag{8}$$

Based on the above equation, the difference in the expected reward between the new and old policies can be represented as

$$\begin{aligned} J(\theta') - J(\theta) &= \mathbb{E}_{s_0} [V^{\pi_{\theta'}}(s_0)] - \mathbb{E}_{s_0} [V^{\pi_\theta}(s_0)] \\ &= \frac{1}{1 - \gamma} \mathbb{E}_{s \sim \nu^{\pi_{\theta'}}} \mathbb{E}_{a \sim \pi_{\theta'}(\cdot | s)} [A^{\pi_\theta}(s, a)] \end{aligned} \tag{9}$$

Therefore, learning the optimal policy π_θ is equivalent to maximizing the bonus optimization object:

$$L_\theta(\theta') = J(\theta) + \mathbb{E}_{s \sim \nu^{\pi_{\theta'}}} \mathbb{E}_{a \sim \pi_{\theta'}(\cdot | s)} [A^{\pi_\theta}(s, a)] \tag{10}$$

When the old and new policies are very close, the optimization objective can be substituted with

$$\begin{aligned} L_{\theta}(\theta') &= J(\theta) + \mathbb{E}_{s \sim \nu^{\pi_{\theta}}} \mathbb{E}_{a \sim \pi_{\theta'}(\cdot|s)} [A^{\pi_{\theta}}(s, a)] \\ &= J(\theta) + \mathbb{E}_{s \sim \nu^{\pi_{\theta}}} \mathbb{E}_{a \sim \pi_{\theta}(\cdot|s)} \left[\frac{\pi_{\theta'}(a|s)}{\pi_{\theta}(a|s)} A^{\pi_{\theta}}(s, a) \right] \end{aligned} \quad (11)$$

To ensure that the new and old policies are sufficiently close, TRPO uses KL divergence to measure the distance between strategies. Then, the optimization formula can be expressed as

$$\begin{aligned} \max_{\theta'} L_{\theta}(\theta') \\ \text{s.t. } \mathbb{E}_{s \sim \nu^{\pi_{\theta}}} [D_{KL}(\pi_{\theta}(\cdot|s), \pi_{\theta'}(\cdot|s))] \leq \delta \end{aligned} \quad (12)$$

Performing Taylor expansions on the objective and constraint to a leading order around θ_k , the optimization formula can be expressed as

$$\begin{aligned} \theta_{k+1} &= \arg \max_{\theta'} g^T(\theta' - \theta_k) \\ \text{s.t. } \frac{1}{2}(\theta' - \theta_k)^T H(\theta' - \theta_k) &\leq \delta \end{aligned} \quad (13)$$

where g represents the gradient of the objective function with respect to the policy parameters θ' and H denotes the Hessian matrix of the KL divergence between the new and old policies. The expressions for g and H are given as follows:

$$g = \nabla_{\theta'} \mathbb{E}_{s \sim \nu^{\pi_{\theta}}} \mathbb{E}_{a \sim \pi_{\theta}(\cdot|s)} \left[\frac{\pi_{\theta'}(a|s)}{\pi_{\theta}(a|s)} A^{\pi_{\theta}}(a|s) \right] \quad (14)$$

$$H = H \left(\mathbb{E}_{s \sim \nu^{\pi_{\theta}}} \left[D_{KL} \left(\pi_{\theta}(\cdot|s), \pi_{\theta'}(\cdot|s) \right) \right] \right) \quad (15)$$

This approximate problem can be analytically solved by the methods of Lagrangian duality, yielding the following solution:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \quad (16)$$

Because the TRPO algorithm approximates the objective function and the constraint conditions with first-order and second-order approximations, respectively, during the optimization process, it does not achieve an exact solution. Consequently, the new policy may not necessarily outperform the old one or may fail to satisfy the KL divergence constraint. Therefore, TRPO performs a linear search at the end of each iteration to ensure that a new policy that meets the conditions is found. The update formula of policy parameters θ can be expressed as

$$\theta_{k+1} = \theta_k + \omega^i \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \quad (17)$$

where $\omega \in (0, 1)$ is the backtracking coefficient.

3. Position Control Policy Training of the PCM Based on MAML Meta-Reinforcement Learning

3.1. The Structure of the PCM

The PCM consists of two sections, and each section adopts the same layout. Each section of the manipulator is composed of three contractile pneumatic muscles, which are distributed at 120° . The PCM is shown in Figure 1. The overall length of the PCM is 830 mm, with each section having a length of 415 mm and a radius of 37.5 mm. Each section of the PCM has the ability to bend in different directions. PCM is driven by pneumatic muscles

by adjusting the air pressure of the pneumatic muscles to control the length changes of the pneumatic muscles. The length changes of the pneumatic muscles can cause the length of the manipulator to change and lead to local bending of the manipulator. By changing the lengths of the pneumatic muscles, the direction and radius of bending of the manipulator can be controlled to drive the manipulator to perform movements such as extension, deflection, and bending.

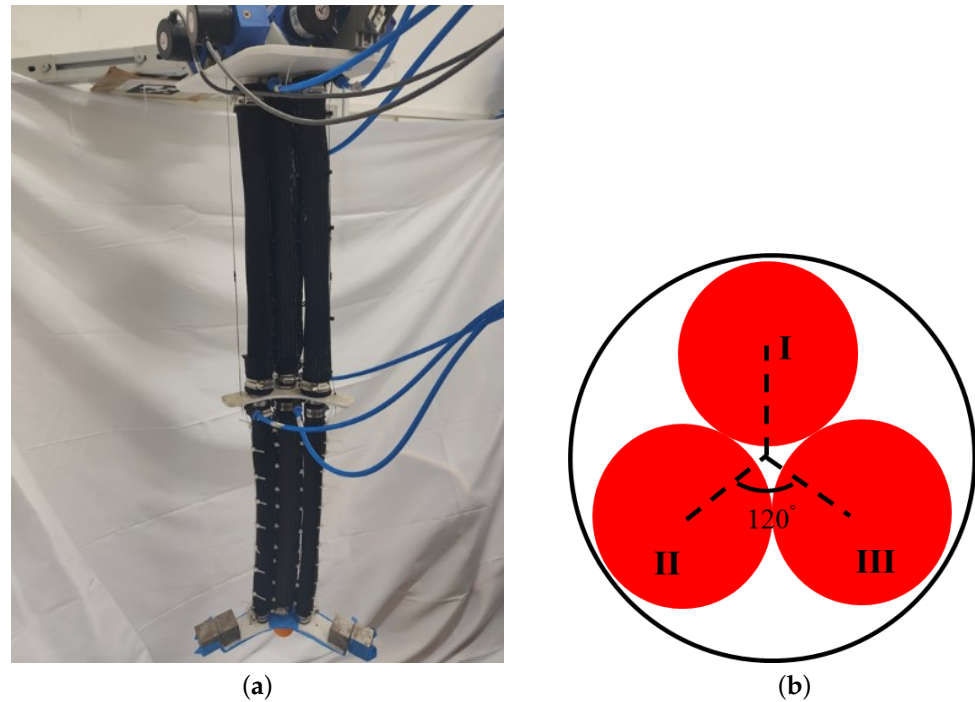


Figure 1. The structure of the PCM. (a) The PCM; (b) the cross-section of the PCM.

The task of end position control for the PCM involves randomly generating a target position within the workspace and controlling the pressure of each pneumatic muscle to drive the PCM to achieve the target position. The motion range of the PCM along the x -axis is $[-200, 200]$ mm; along the y -axis, it is $[-200, 200]$ mm; and, along the z -axis, it is $[-100, 100]$ mm. The movement range of the end-effector along the x -axis is set to $[-200, 200]$ mm; along the y -axis, it is set to $[-200, 200]$ mm; and, along the z -axis, it is set to $[-100, 100]$ mm. For end position control, each target position corresponds to a task, and these tasks collectively form a task distribution. The task distribution $p(\mathcal{T})$ of the PCM is a uniform distribution within its range of motion.

3.2. Meta-Reinforcement Learning Modeling and Policy Network Structure Design

State space (s). State information is crucial for making action decisions. The state of the continuum manipulator is determined by the current pressure values of each pneumatic muscle. Therefore, the state space is composed of the current pressure values of each pneumatic muscle, denoted as $s = [u_{11}, u_{12}, u_{13}, u_{21}, u_{22}, u_{23}]$ with a dimension of 6.

Action space (a). The action space encompasses the entire set of actions available to the agent. Each pneumatic muscle of the PCM can be independently pressurized and depressurized. So, the pressure increments of each pneumatic muscle Δu can be used as actions, that is $a = [\Delta u_{11}, \Delta u_{12}, \Delta u_{13}, \Delta u_{21}, \Delta u_{22}, \Delta u_{23}]$, with a dimension of 6.

Reward function (r). The reward function plays a critical role in reinforcement learning training. It is primarily used to guide the agent's behavior. For actions that align with the desired behavior, the reward function provides a positive reward, while for actions that do not, it provides a negative reward. Thus, the design of the reward function directly impacts the learning speed, operational efficiency, and the final control performance of the trained

model. As the objective of this paper is to control the end-effector of the PCM to reach a specified position, the negative of the difference between the current end position and the target position can be used as the reward function to encourage the policy to control the end position of the PCM as closely as possible to the target position. So, the reward function can be expressed as

$$r = -\left[(x - x_{target})^2 + (y - y_{target})^2 + (z - z_{target})^2\right] \tag{18}$$

where (x, y, z) is end position of the PCM and $(x_{target}, y_{target}, z_{target})$ is the target position.

Structure of policy network. For the task of the end position control of the PCM, the design of the policy network structure is based on the action space and state space. The policy network learns the mapping from the current state to the optimal action. In the REINFORCE algorithm, the input to the policy network is the state of the agent at the current time, and the output is the mean and variance of the normal distribution of the action taken by the agent. The policy is represented by a four-layer fully connected network, with 256 neurons in the middle two layers, and each layer uses the tanh activation function. The Adam optimizer is employed to compute the policy gradient and update the network weights.

The policy network outputs a Gaussian distribution of the actions taken by the agent. Sampling from this Gaussian distribution yields the specific action taken by the agent in the current state, along with the probability of taking that action.

4. Simulation Analysis

4.1. Simulation Environment

For simulation, the dynamic model of the PCM is used. The dynamic model of the PCM can be expressed as the mapping between the current time control input u_t , the previous control inputs u_{t-1}, \dots, u_{t-T} , the position at the current time x_t , and the position at the previous time x_{t-1}, \dots, x_{t-T} . Specifically, it can be expressed as

$$x_{t+1} = f(u_t, u_{t-1}, \dots, u_{t-T}, x_t, x_{t-1}, \dots, x_{t-T}) \tag{19}$$

where T is the horizon length and f represents the dynamics model of the PCM.

This paper uses an LSTM network to model the dynamics of the PCM. The input of the LSTM network dynamic model is the pressure value of each pneumatic muscle, and the output is the end-effector position of the PCM.

Figure 2 illustrates the architecture of the proposed LSTM network for the dynamics model of the PCM. The network comprises an LSTM layer and a fully connected layer. The LSTM layer has 200 neurons, while the fully connected layer has three neurons, and the horizon length $T = 3$.

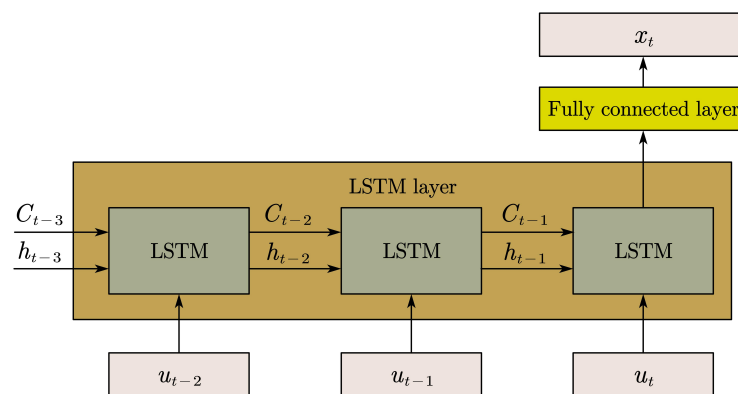


Figure 2. The architecture of the dynamics model using LSTM network.

The dataset is sampled using pseudorandom motor babbling, yielding 21,000 data points. Normalizing the data, 80% is used for training, with the remainder serving as a validation set. The LSTM network is trained using the Adam optimization algorithm, with the root mean square error (RMSE) as the loss function. The initial learning rate is set to 0.1, the learning rate decay factor is 0.5, the learning rate decay period is 5, the batch size is 32, and the number of training iterations is 30. The change in RMSE during training is shown in Figure 3.

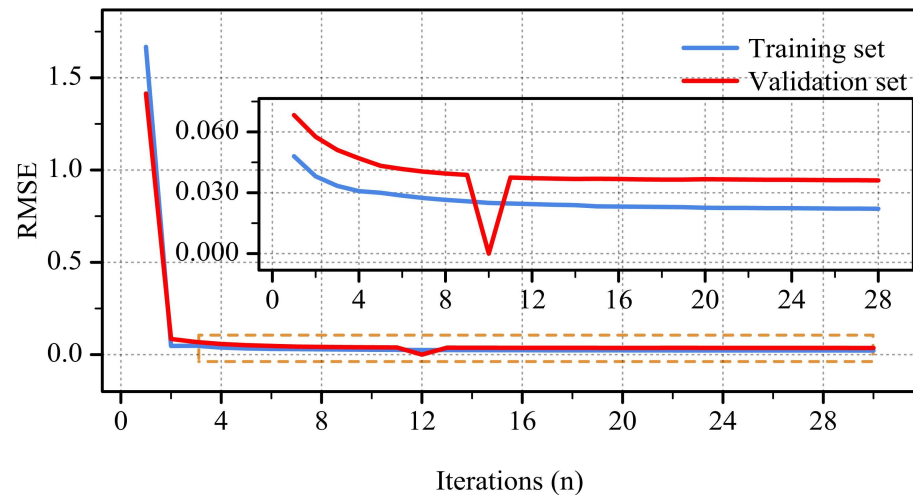


Figure 3. The training result of the dynamics model.

As shown in Figure 3, on the training set, the RMSE converges to 0.02 after training. On the validation set, the RMSE converges to 0.03, and the model's prediction accuracy is 91.2%. Therefore, the LSTM model is capable of representing the dynamics model of the PCM.

4.2. Training Overview of MAML Meta-Reinforcement Learning

To verify the adaptive capabilities of MAML meta-reinforcement learning in end-position control tasks for PCM, this section investigates the generalization performance of the PCM control policy trained by MAML meta-reinforcement learning through simulations.

For reinforcement learning, generalization refers to the number of steps an agent needs to take to achieve good performance in a new task. It includes two evaluation metrics, and one is the expected reward of the agent in the new task. The expected reward reflects the overall reward that the agent receives in new tasks. The higher the expected reward that the agent receives in new tasks, the better its performance in new tasks. The other metric is the adaptation step number, which refers to how many steps the agent needs to interact with the environment in order to achieve a good performance in the new task. The number of adaptation steps reflects the ability of the agent to adapt to new tasks. The fewer the adaptation steps, the faster the agent adapts to the new tasks.

Different target positions are sampled from the task distribution $p(\mathcal{T})$ as training tasks, and the agent interacts with the environment based on these different tasks. In the experiment, the agent is set to interact with the environment to generate 20 trajectories under each task, with each trajectory having a length of 10. Within the inner loop, each batch of tasks undergoes only one gradient update during the training phase. The hyperparameters of the MAML meta-reinforcement learning algorithm during the training process are set as shown in Table 1.

The training is carried out on a desktop computer with an operating system of Windows 10, the CPU is Intel (R) Core (TM) i7-6700 CPU @ 3.40GHz, and the memory is 16 GB.

Table 1. Hyperparameters of MAML meta-reinforcement learning.

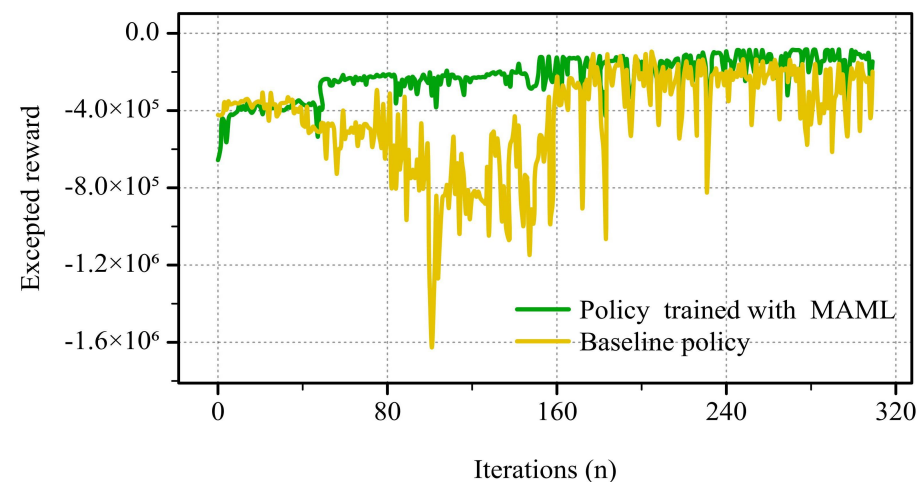
Parameters	Values
Learning rate α	1×10^{-14}
Size of the trust-region δ	1×10^{-2}
Conjugate gradient iterations	10
Maximum line search iterations	15
Line search backtrack coefficient	0.8
Inner loop task samples	20
Number of trajectories collected per task	20
Inner loop optimization iterations	1
Outer loop task samples	20
Outer loop optimization iterations	1000

4.3. Simulation Result

Using the policy initialized randomly as a baseline policy, compare the generalization ability on new tasks with that of the policy initialized using MAML meta-reinforcement learning. In order to verify the effectiveness of MAML meta-reinforcement learning, this paper samples the following two tasks from the task distribution for analysis.

4.3.1. Task 1

Take the task of controlling PCM to reach the target point (0, 80, 740) mm as the new task. To evaluate the generalization performance, both the policy initialized using the MAML meta-reinforcement learning and the baseline policy are trained to adapt to new tasks using the REINFORCE algorithm, and their generalization performances are compared. The training processes for both policies employ the same hyperparameters. The training result is shown in Figure 4.

**Figure 4.** Result of Task 1.

As illustrated in Figure 4, when faced with a new task, the policy initialized with MAML meta-reinforcement learning reaches convergence in the excepted reward after 60 updates with a training time of 5 min. The baseline policy reaches convergence in the excepted reward after 180 updates with a training time of 15 min. By comparison, it is evident that the MAML meta-reinforcement learning algorithm has learned effective initial weights for the policy network from historical experiences. These initial weights enable the policy to adapt to new tasks with minimal updates, improving the adaptive capability for end position control of the PCM.

After the training of the policy initialized by MAML meta-reinforcement learning is completed, we use the trained policy to control the PCM. The position changes and the position error changes of the PCM are shown in Figures 5 and 6, respectively.

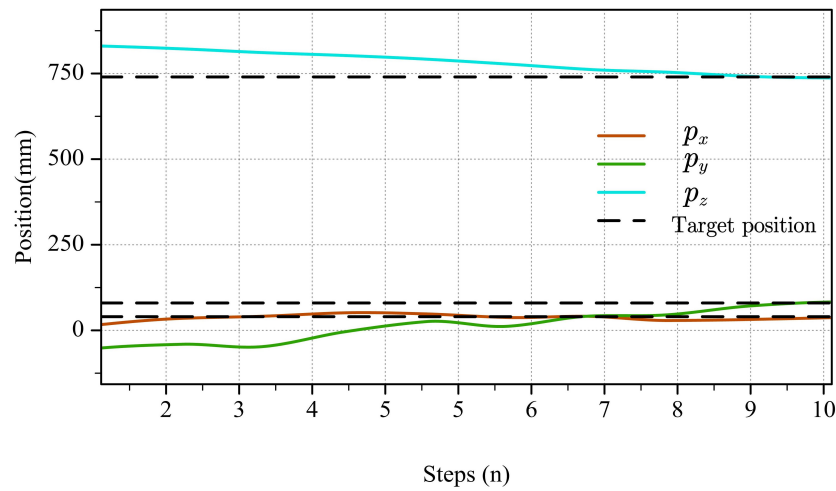


Figure 5. The position changes of the PCM after policy adaptation in Task 1.

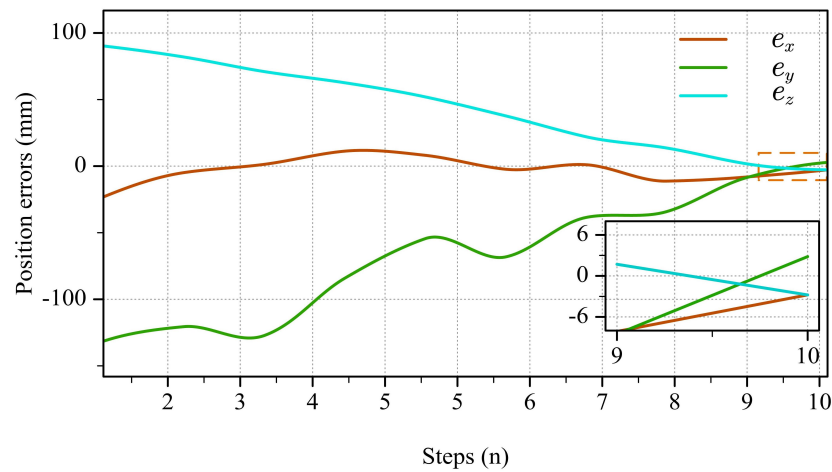


Figure 6. The position error changes of the PCM after policy adaptation in Task 1.

The position error is (2.77, −2.82, −2.78) mm, and the maximum error is within 1.4%. Thus, the policy initialized with MAML meta-reinforcement learning can effectively control the PCM to reach the new target point after training.

To compare the performance with other reinforcement learning algorithms, the control policies for PCM are trained using the PPO algorithm, the Actor–Critic algorithm, and the PILCO algorithm. The hyperparameters of the PPO algorithm are as follows: the clipping parameter $\epsilon = 0.2$, the learning rate for the policy network $l_a = 1 \times 10^{-4}$, and the learning rate for the value network $l_c = 5 \times 10^{-3}$. Both the policy network and the value network have one hidden layer, and the number of neurons in the hidden layer is 256. The hyperparameters of the Actor–Critic algorithm are as follows: the learning rate for the policy network $l_a = 1 \times 10^{-4}$, and the learning rate for the value network $l_c = 5 \times 10^{-3}$. Both the policy network and the value network have one hidden layer, and the number of neurons in the hidden layer is 256. After the training of the policy, we use the trained policy to control the PCM. The position changes and position error changes of the PCM are shown in Figures 7 and 8, respectively.

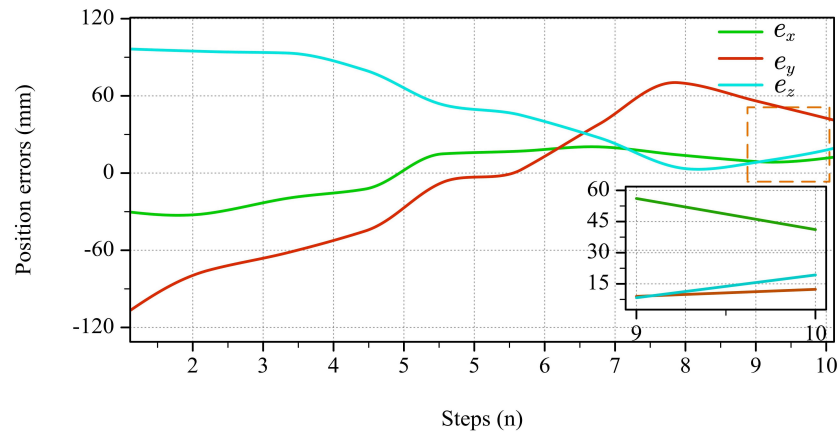


Figure 7. The position changes of the PCM after training by PPO in Task 1.

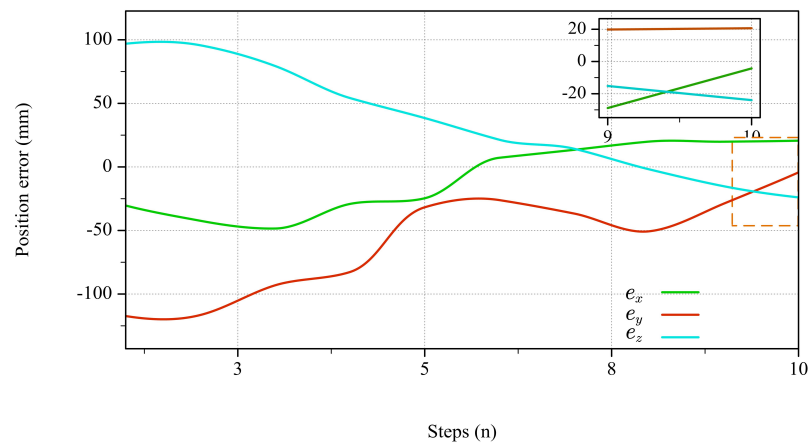


Figure 8. The position error changes of the PCM after training by Actor-Critic in Task 1.

The training times and errors of these algorithms are shown in Table 2.

Table 2. Comparison of training results in task 1.

Algorithm	Training Time (min)	Maximum Absolute Error (mm)
MAML meta-reinforcement learning	5	2.82
PPO	28	41.12
Actor-Critic	17	23.97
PILCO	47	4.6

From Table 2, it can be seen that the policy initialized with MAML meta-reinforcement learning requires the shortest training time and achieves higher control precision during new task training. Compared to other algorithms, this approach enhances both training speed and control accuracy.

4.3.2. Task 2

Take the task of controlling PCM to reach the target point (22, −166, 736) mm as the new task. To evaluate the generalization performance, both the policy initialized using the MAML meta-reinforcement learning and the baseline policy are trained to adapt to new tasks using the REINFORCE algorithm, and their generalization performances are compared. The training processes for both policies employ the same hyperparameters. The training result is shown in Figure 9.

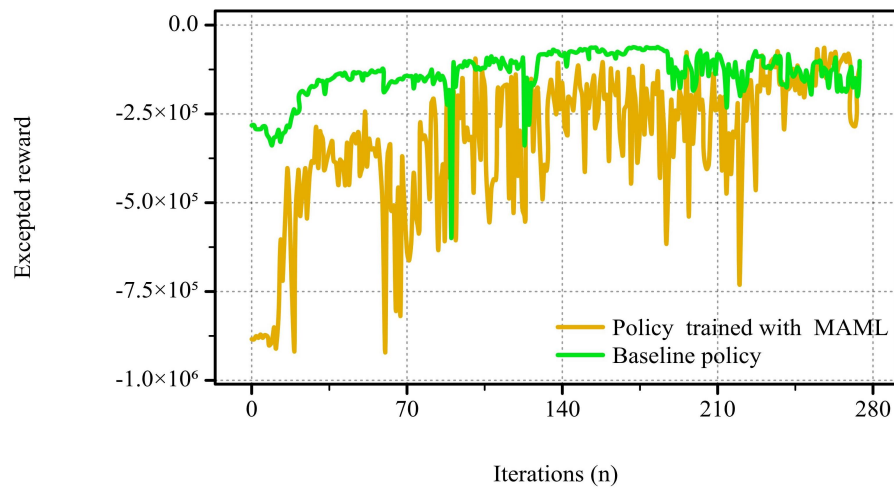


Figure 9. Result of Task 2.

As illustrated in Figure 4, when faced with a new task, the policy initialized with MAML meta-reinforcement learning reaches convergence in the excepted reward after 60 updates with a training time of 5 min. The baseline policy reaches convergence in the excepted reward after 180 updates with a training time of 15 min. By comparison, it is evident that the MAML meta-reinforcement learning algorithm has learned effective initial weights for the policy network from historical experiences. These initial weights enable the policy to adapt to new tasks with minimal updates, improving the adaptive capability for end-position control of the PCM.

After the training of the policy initialized by MAML meta-reinforcement learning is completed, we use the trained policy to control the PCM. The position changes and position error changes of the PCM are shown in Figures 10 and 11, respectively.

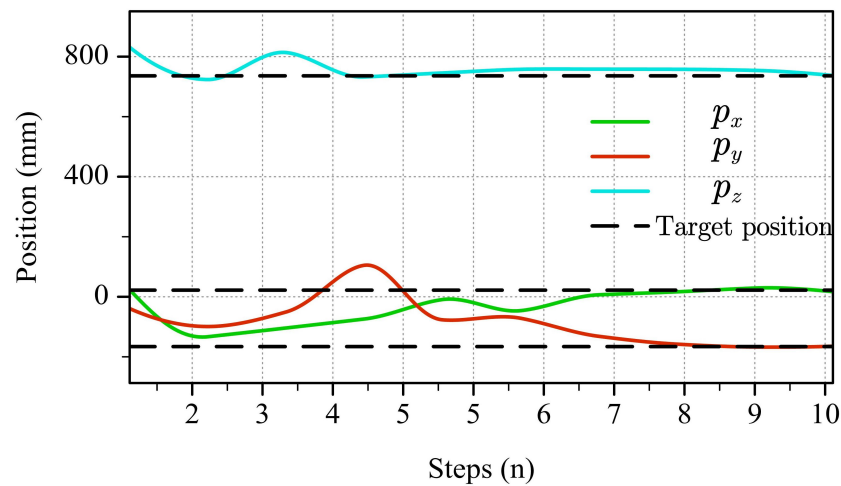


Figure 10. The position changes of the PCM after policy adaptation in Task 2.

The position error is (2.77, −2.82, −2.78) mm, and the maximum error is within 1.4%. Thus, the policy initialized with MAML meta-reinforcement learning can effectively control the PCM to reach the new target point after training.

To compare the performance with other reinforcement learning algorithms, the control policies for PCM are trained using the PPO algorithm, the Actor–Critic algorithm, and the PILCO algorithm. The hyperparameters of the PPO algorithm are as follows: the clipping parameter $\epsilon = 0.2$, the learning rate for the policy network $l_a = 1 \times 10^{-4}$, and the learning rate for the value network $l_c = 5 \times 10^{-3}$. Both the policy network and the value network have one hidden layer, and the number of neurons in the hidden layer is 256. The

hyperparameters of the Actor–Critic algorithm are as follows: the learning rate for the policy network $l_a = 1 \times 10^{-4}$, and the learning rate for the value network $l_c = 5 \times 10^{-3}$. Both the policy network and the value network have one hidden layer, and the number of neurons in the hidden layer is 256. After the training of the policy, we use the trained policy to control the PCM. The position changes and position error changes of the PCM are shown in Figures 12 and 13, respectively.

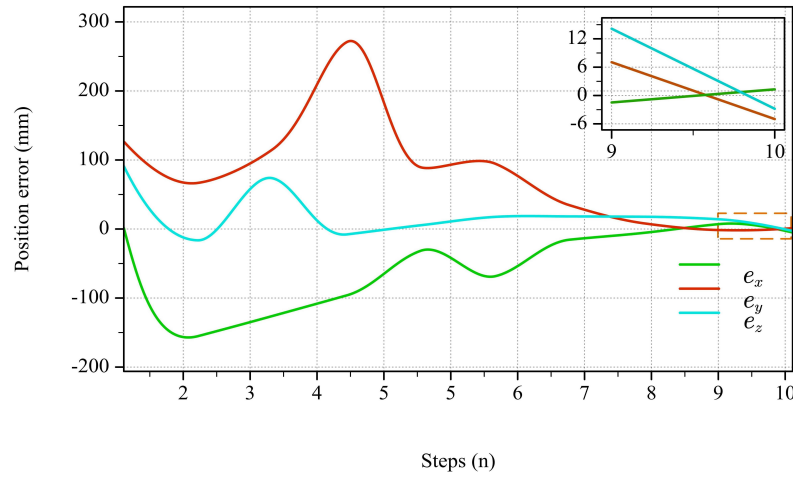


Figure 11. The position error changes of the PCM after policy adaptation in Task 2.

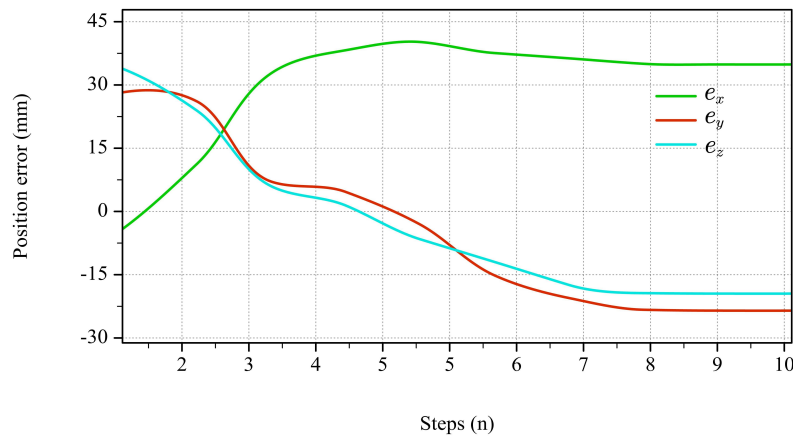


Figure 12. The position changes of the PCM after training by PPO in Task 2.

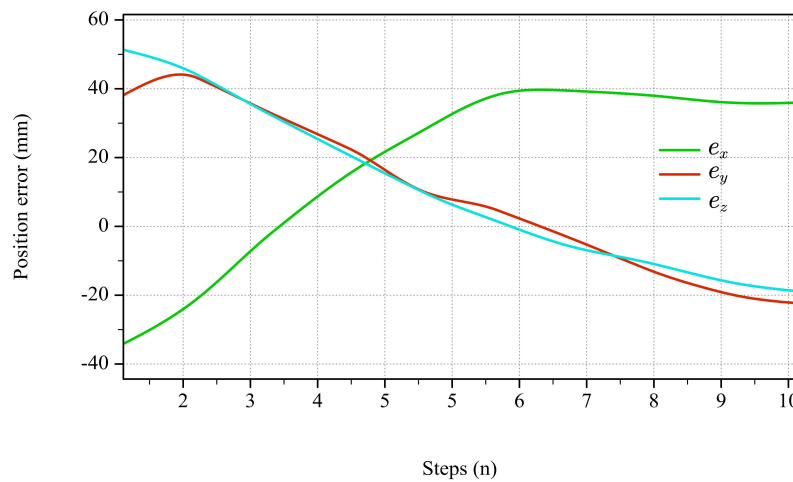


Figure 13. The position error changes of the PCM after training by Actor–Critic in Task 2.

The training times and errors of these algorithms are shown in Table 3.

Table 3. Comparison of training results in task 2.

Algorithm	Training Time (min)	Maximum Absolute Error (mm)
MAML meta-reinforcement learning	5	4.9
PPO	20	34.8
Actor-Critic	25	25.9
PILCO	47	3.9

From Table 3, it can be seen that the policy initialized with MAML meta-reinforcement learning requires the shortest training time and achieves higher control precision during new task training. Compared to other algorithms, this approach enhances both training speed and control accuracy.

5. Experiment

5.1. Experimental Platform

Take the two-section PCM as the experimental object, The experimental platform is shown in Figures 14 and 15. The setup included a power supply, the PCM, a camera, proportional valves, an I/O module, a PC, and an air compressor. The camera captured the position of the manipulator’s end effector, and the PC output control signals to the PCM based on this position. After receiving the control signals from the PC, the I/O module would drive the proportional valves to adjust the pressure of the pneumatic muscles, thereby driving the end effector of the pneumatic continuum manipulator to move.

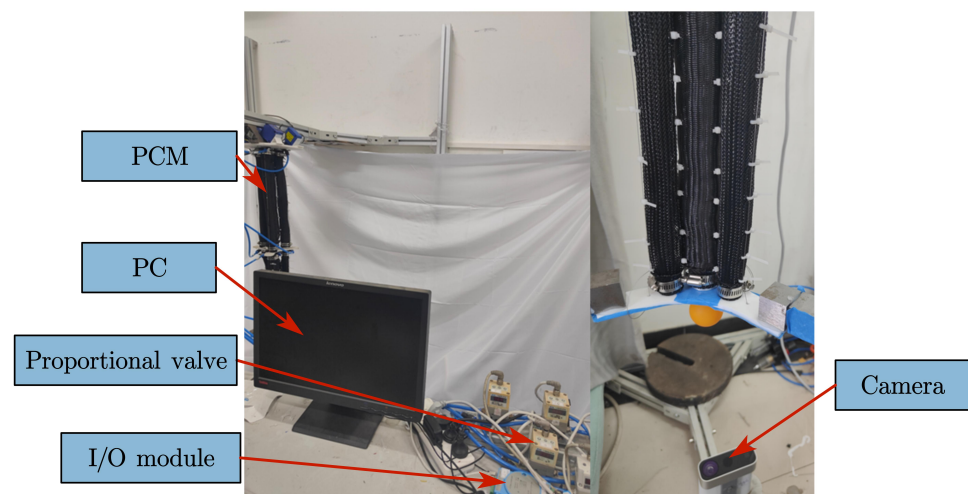


Figure 14. Experiment setup.

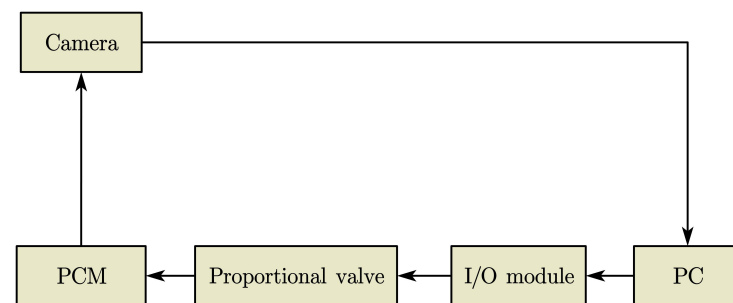


Figure 15. Schematic of experimental setup.

5.2. Experimental Analysis

In the real world, the same initial and target positions as in Task 1 in the simulation were selected, that is, to control the PCM to reach the target point (40, 80, 740) mm.

Employing the sim-to-real transfer approach based on policy fine-tuning, the control policy was fine-tuned on the actual PCM. After fine-tuning, we used the method of sim-to-real transfer based on policy fine-tuning to retrain the control policy on the PCM. After training, the position changes and position error changes of the PCM are shown in Figures 16 and 17, respectively.

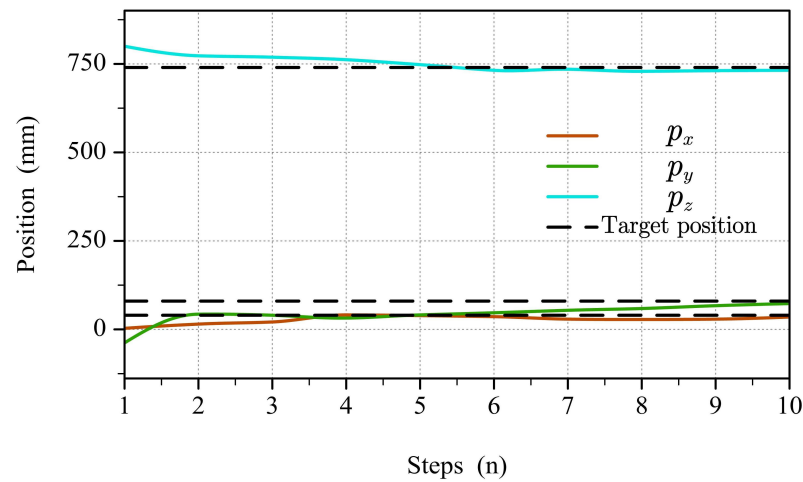


Figure 16. The position changes of the real PCM.

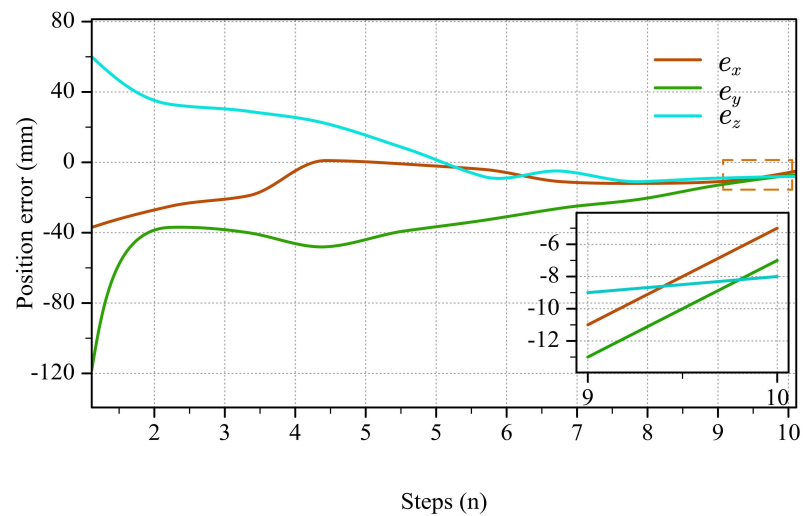


Figure 17. The position error changes of the real PCM.

As shown in Figures 16 and 17, the policy can control the PCM to reach the target position, with maximum errors of 8 mm along the X-axis, Y-axis, and Z-axis. The maximum error is within 4%.

6. Conclusions

This paper applies meta-reinforcement learning methods to the position control of the PCM, using MAML meta-reinforcement learning to train the position control policy of the PCM. MAML meta-reinforcement learning dynamically adjusts control policy to adapt to different tasks by learning general initialization parameters of policy, thereby enhancing the adaptability of the policy to new control tasks. Finally, based on the experimental platform, the adaptive ability of MAML meta-reinforcement learning was verified. The

results showed that the control policy trained by the MAML meta-reinforcement learning algorithm could complete the control task of the PCM with minimal training when facing new tasks, improving the adaptability of the control policy.

Author Contributions: Validation, Q.C.; Resources, Y.Z.; Writing—original draft, L.H.; Writing—review & editing, H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China under Grant Number 62073063 and National Natural Science Foundation of China for Youths under Grant 62103090.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. You, X.; Zhang, Y.; Chen, X.; Liu, X.; Wang, Z.; Jiang, H.; Chen, X. Model-free control for soft manipulators based on reinforcement learning. In Proceedings of the 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 2909–2915.
2. Satheeshbabu, S.; Uppalapati, X.K.; Chowdhary, G.; Krishnan, G. Open loop position control of soft continuum arm using deep reinforcement learning. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 5133–5139.
3. Satheeshbabu, S.; Uppalapati, X.K.; Fu, T.; Krishnan, G. Continuous control of a soft continuum arm using deep reinforcement learning. In Proceedings of the 2020 3rd IEEE International Conference on Soft Robotics (RoboSoft), New Haven, CO, USA, 15 May–15 July 2020; pp. 497–503.
4. Li, Y.; Wang, X.; kwok, K. Towards adaptive continuous control of soft robotic manipulator using reinforcement learning. In Proceedings of the 2022 IEEE/RSJ international conference on intelligent robots and systems (IROS), Kyoto, Japan, 23–27 October 2022; pp. 7074–7081.
5. Centurelli, A.; Rizzo, A.; Tolu, S.; Laschi, C. Closed-loop dynamic control of a soft manipulator using deep reinforcement learning. *IEEE Robot. Autom. Lett.* **2022**, *7*, 4741–4748. [[CrossRef](#)]
6. Pertsch, K.; Lee, Y.; Lim, J.J. Accelerating reinforcement learning with learned skill priors. *arXiv* **2020**, arXiv:2010.11944.
7. Rusu, A.A.; Colmenarejo, C.G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; Hadsell, R. Policy distillation. *arXiv* **2015**, arXiv:1511.06295.
8. Zhang, Y.; Yang, Q. A survey on multi-task learning. *IEEE Trans. Knowl. Data Eng.* **2021**, *34*, 5586–5609. [[CrossRef](#)]
9. Schweighofer, N.; Doya, K. Meta-learning in reinforcement learning. *Neural Netw.* **2021**, *34*, 5586–5609. [[CrossRef](#)] [[PubMed](#)]
10. Duan, Y.; Schulman, J.; Chen, X.; Bartlett, P.L.; Sutskever, I.; Abbeel, P. RL^2 : fast reinforcement learning via slow reinforcement learning. *arXiv* **2016**, arXiv:1611.02779.
11. Mishra, N.; Rohaninejad, M.; Chen, X.; Abbeel, P. A simple neural attentive meta-learner. *arXiv* **2017**, arXiv:1707.03141.
12. Parisotto, E.; Ghosh, S.; Yalamanchi, S.B.; Chinnabireddy, V.; Wu, Y.; Salakhutdinov, R. Concurrent meta reinforcement learning. *arXiv* **2019**, arXiv:1903.02710.
13. Rakelly, K.; Zhou, A.; Finn, C.; Levine, S.; Quillen, D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 5331–5340.
14. Zhang, J.; Wang, J.; Hu, H.; Chen, T.; Chen, Y.; Fan, C.; Zhang, C. MetaCURE: meta reinforcement learning with empowerment-driven exploration. In Proceedings of the 38th International Conference on Machine Learning, Virtual, 18–24 July 2021; pp. 12600–12610.
15. Fakoor, R.; Chaudhari, P.; Soatto, S.; Smola, A.J. Meta-Q-learning. *arXiv* **2019**, arXiv:1910.00125.
16. Hu, S. Research of Meta-Reinforcement Learning and Its Application in Multi-Legged Robots. Master's Thesis, China University of Mining and Technology, Xuzhou, China, 2023.
17. Finn, C.; Abbeel, P.; Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In Proceedings of the 34th International conference on machine learning, Sydney, Australia, 6–11 August 2017; pp. 1126–1135.
18. Song, X.; Gao, W.; Yang, Y.; Choromanski, K.; Pacchiano, A.; Tang, Y. ES-MAML: simple hessian-free meta learning. *arXiv* **2019**, arXiv:1910.01215.
19. Nichol, A.; Schulman, J. Reptile: a scalable metalearning algorithm. *arXiv* **2018**, arXiv:1803.02999.

20. Gupta, A.; Mendonca, R.; Liu, Y.X.; Abbeel, P.; Levine, S. Meta-reinforcement learning of structured exploration strategies. In Proceedings of the 32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, QC, Canada, 3–8 December 2018; pp. 5307–5316.
21. Liu, H.; Socher, R.; Xiong, C. Taming MAML: efficient unbiased meta-reinforcement learning. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 4061–4071.
22. Li, M.; Xu, G.; Gao, X.; Tan, C. A reaching skill learning method of manipulators based on meta-Q-learning and DDPG. *J. Nanjing Univ. Posts Telecommun. (Nat. Sci. Ed.)* **2023**, *43*, 96–103.
23. Hao, T. Research on Meta-Optimizer and Meta-Reinforcement Learning Based on Bi-Level Optimization Meta-Learning. Master's Thesis, Shandong University, Jina, China, 2022.
24. Schoettler, G.; Nair, A.; Ojea, J.A.; Levine, S.; Solowjow, E. Meta-reinforcement learning for robotic industrial insertion tasks. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020–24 January 2021; pp. 9728–9735.
25. Peng, K. Robot Motion Control Method Based on Particle Swarm Optimization and Meta-Reinforcement Learning. Master's Thesis, Yangzhou University, Yangzhou, China, 2022.
26. Wang, L.; Zhang, Y.; Zhu, D.; Coleman, S.; Kerr, D. Supervised meta-reinforcement learning with trajectory optimization for manipulation tasks. *IEEE Trans. Cognit. Dev. Syst.* **2023**, *16*, 681–691. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.