Meta Reinforcement Learning of Locomotion Policy for Quadruped Robots With Motor Stuck

Ci Chen, Chao Li, Haojian Lu, *Member, IEEE*, Yue Wang, *Member, IEEE*, and Rong Xiong, *Senior Member, IEEE*

***Abstract*— Significant progress has been made in enhancing the motion capabilities of quadruped robots in unstructured environments due to advancements in hardware and control algorithms. However, limited research has been conducted on the fault-tolerant control of quadruped robots, which is crucial for their operation in remote or extreme environments like disaster sites. In this paper, we primarily focus on fault-tolerant strategies for common joint-stuck situations. By leveraging the static stability of quadruped robots, it becomes possible to adjust their control policies and enable them to continue following predetermined trajectories. We introduce a contextual meta-reinforcement learning (Meta-RL) method to design fault-tolerant policies. This method infers task-related latent vectors from the context to assist in training the policy network, ensuring both conciseness and optimality in various situations. Additionally, to expedite algorithm training, we propose a reference action generator (RAG). To validate the proposed algorithm, extensive simulations and physical experiments are conducted. The results demonstrate that our method allows the robot to maintain its trajectory even when faced with motor locking. Furthermore, our method outperforms all baseline algorithms, highlighting its superiority in terms of fault tolerance.**

***Note to Practitioners*—The motivation of this article is to provide fault-tolerant policies for quadruped robots, specifically referring to the policies for joint-stuck situations. Previous fault-tolerant strategies either require individually designing control strategies for each joint stuck task, which brings a significant workload to designers, or adopting a unified strategy that cannot provide the optimal strategy for each task. In this article, we utilize the Meta-RL method to handle the joint stuck issue in robots for the first time. By combining the context encoder and RAG, we can provide more suitable policies for various motor-stuck tasks. Both the simulation and physical experiments validate the effectiveness and applicability of this method.**

***Index Terms*— Meta reinforcement learning, quadruped robots, fault tolerance.**

Manuscript received 8 May 2024; accepted 29 June 2024. This article was recommended for publication by Associate Editor W. Zhang and Editor P. Rocco upon evaluation of the reviewers' comments. This work was supported in part by the National Science and Technology Major Project of China under Grant 2021ZD0114504, in part by the National Nature Science Foundation of China under Grant 62373322 and Grant 62303407, in part by Zhejiang Provincial Natural Science Foundation of China under Grant LD22E050007, and in part by the Innovation and Development Special Fund of Hangzhou Chengxi Sci-Tech Innovation Corridor. *(Corresponding author: Yue Wang.)*

Ci Chen, Haojian Lu, Yue Wang, and Rong Xiong are with the State Key Laboratory of Industrial Control and Technology and the Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China (e-mail: wangyue@iipc.zju.edu.cn).

Chao Li is with DeepRobotics Company, Hangzhou 310058, China.

Data is available on-line at: https://github.com/chenci107/MetaRL_for_Quadruped.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TASE.2024.3424328.

Digital Object Identifier 10.1109/TASE.2024.3424328

I. Introduction

QUADRUPED robots are extensively utilized in various scenarios, including search and rescue operations, disaster response efforts, and routine inspections, primarily due to their exceptional motion capabilities [1], [2]. However, given the uncontrolled environments in which they operate, repairing robots' faults becomes an arduous task. Hence, it is crucial to develop an algorithm that can equip quadruped robots with fault tolerance [3], [4], [5], [6]. Motor locks are a common issue in robot failures and can result from various causes, such as mechanical structure damage, hardware communication interruption, driver failure, or motor overload break [7]. When a joint becomes stuck, the robot's degree of actuator (DOA) is reduced, significantly impacting legged locomotion in terms of mobility and stability. This could lead to the robot's inability to continue performing tasks or return to a safe position. However, quadruped robots possess redundancy, which enables them to tolerate faults. By effectively adjusting the control policy, quadruped robots can maintain a normal walking pattern.

It is a challenging task to equip a quadruped robot with fault-tolerant capability. Traditional methods [6], [8], [9], [10] involve analyzing the leg's workspace, designing fault-tolerant gaits, and subsequently performing kinematic inverse solutions. In a recent study, [7] combines fault-tolerant control with whole-body control (WBC) to achieve stable and continuous forward walking of a quadruped robot. However, these methods require extensive expertise and entail a laborious manual tuning process. Additionally, such methods necessitate designing controllers with different parameters for various joint stuck situations, placing a significant workload on the designer.

Fortunately, the advancement of deep reinforcement learning (RL) allows for the design of fault-tolerance strategies without the need for extensive manual intervention [11], [12], [13], [14]. The RL methods enable learning through trial and error, allowing the agent to autonomously discover effective strategies. Generally, conventional RL aims to maximize rewards for specific tasks, with learned policies

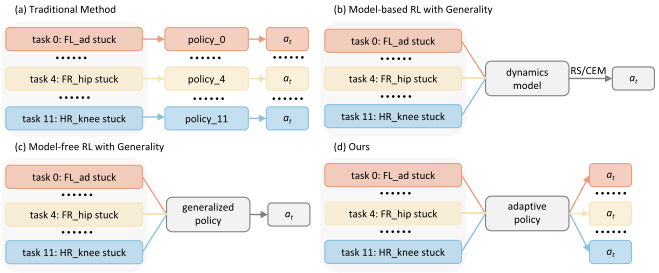


Fig. 1. Comparisons of different fault-tolerant methods. (a) Traditional methods involve designing separate controllers with different parameters for each motor stuck situation, which can be cumbersome. (b) Model-based RL methods are challenging to guarantee optimality across different tasks and meet the real-time requirements of the quadruped robot system. (c) Model-free RL methods with generality employ a single controller to handle various motor-stuck situations but sacrifice optimality in different scenarios. (d) The proposed method utilizes the context-based Meta-RL method, offering a concise yet optimal solution. Additionally, it is easier to deploy on real robots.

performing well only within those tasks. However, different joint failures in quadruped robots can lead to various changes in the robot's dynamics, resulting in different state transition probabilities, which are considered distinct tasks. To ensure the adaptability of control strategies across a variety of tasks, some model-based RL methods [11], [12], [13], [15] train a universal dynamics model using interaction data collected under various scenarios of joint stuck, based on this dynamics model, planning methods are used to determine the next action. However, applying such methods to the twelve-joint quadruped robot presents a challenge. A single dynamics model struggles to provide accurate estimates for each joint stuck scenario, leading to planning algorithms being unable to deliver optimal actions for each task. Additionally, the computational complexity of each planning iteration is very high, making it difficult to apply to the quadruped robots that demand high real-time performance. In addition to model-based RL methods, there are also several studies utilizing model-free RL methods [14], [16], [17] to address the design of fault-tolerance strategies, predominantly employing domain randomization techniques. This involves randomly locking different joints during training, with the aim of learning a generalized control policy capable of managing different joint failures. Similar to model-based methods, the optimality of model-free methods is limited since different joint-stuck scenarios require different optimal control strategies. In summary, the advantages and disadvantages of the aforementioned methods are depicted in Fig. 1.

Unlike conventional RL methods, which are designed to solve single task and require sacrificing optimality to achieve generality across multiple tasks, meta-reinforcement learning (Meta-RL), also known as "Learning to learn", requires models not only to excel at specific tasks but also to acquire the capability to learn, which enables rapid adaptation to new tasks. Specifically, Meta-RL differs from conventional RL in two main aspects. Firstly, the training data originates from a variety of tasks, and the learned strategies are applied to a collection of tasks rather than a single task. Secondly, the process is divided into two phases: the training phase, where a strategy is learned from a set of training tasks, and the adaptation phase, where the strategy is applied to new, unseen tasks after minimal interaction. Meta-RL is inherently suited for handling multi-tasks due to the aforementioned characteristics. In this paper, we formulate the joint stuck scenarios in quadruped robots as a Meta-RL problem. By mimicking humans, agents are encouraged to acquire useful knowledge from a set of training tasks, which then facilitates rapid adaptation to new tasks. Our contributions are as follows.

1) We model the issue of quadruped robot joints stuck as a Meta-RL problem, adopting a context-based method for resolution. Specifically, we design a context encoder to infer information related to joint stuck, providing prior knowledge to assist in the training of the policy network. By incorporating an additional encoder, our method significantly augments the policy's capability to obtain optimal actions tailored to specific scenarios, offering a significant edge over the generality-focused conventional RL methods. Compared to the traditional method of designing a customized strategy for each task, our method is more streamlined. Additionally, we incorporate the concept of curriculum learning into the training process, enabling the learned strategies to handle situations where joints are stuck at different angles.
2) We devise a closed-loop reference action generator (RAG) to facilitate Meta-RL training based on the concept of residual learning. Instead of directly outputting the motor's expected position, Meta-RL performs adjustments based on the outputs from RAG. To address the issue of joint stuck, we design a selection module to identify injured legs, thereby providing the RAG with a more reliable reference signal, and establishing a closed-loop control. This method greatly alleviates the difficulty of Meta-RL learning, accelerates the algorithm's convergence, and elevates performance.
3) A comprehensive series of ablation studies, comparative analysis, and experiments under conditions such as omnidirectional movement and multi-joint stuck are conducted. The results from both simulation and physical experiments demonstrate the feasibility, effectiveness, and advancements of the proposed method.

The rest of this paper is structured as follows: Section II provides a comprehensive review of related works. Our method is detailed in Section III. In Section IV, we present and discuss the experimental results. Finally, Section V concludes the paper and outlines future research directions.

II. Related Works

A. Fault-Tolerant Control in Robotics

Fault-tolerant control of quadruped robots can be classified into two categories: traditional methods and learning-based methods. Traditional methods, as studied in [6], [8], [9], and [10], analyze the workspace of an individual leg with various stuck joints to design gait patterns and parameters for different walking situations of the quadruped robot. However, these methods typically limit the faulty leg to a supportive role, failing to fully exploit its potential by utilizing the available motion space. In contrast, [18] proposes a method

to ensure the quadruped robot's displacement and derives a kinematics solution of posture to utilize the fault leg's workspace. Reference [19] introduces the use of two moving appendages in a pinion arrangement to alter the robot's center of mass (COM) during a fault. Additionally, [7] suggests a fault-tolerant method using an equivalent geometric model to rebuild the workspace of the failed leg. They further optimize the body posture using a nonlinear approximation formula based on this model, which enables smooth robot movement when combined with WBC. However, these methods either require additional mechanical structures or necessitate designing controllers with different parameters for each joint, resulting in significant complexity.

Learning-based methods for fault-tolerant control can be broadly categorized into two categories: model-based RL methods and model-free RL methods. For the first category, the method in [11] trains a prior dynamics model, which, when integrated with recent data, facilitates rapid adaptation to the local context. Reference [13] proposes a trajectory-wise multiple-choice learning method, where a multi-headed dynamics model is learned for dynamics generalization. Similarly, [12] utilizes an ensemble network to train the dynamics model while incorporating future trajectories and damage information to enhance generalization. Once the dynamics model is obtained, these methods rely on model predictive control (MPC) and employ algorithms like random shooting (RS) or cross entropy method (CEM) to select the next action. On the other hand, model-free RL methods, as discussed in [14], introduce an adaptive RL algorithm with dynamics randomization. This algorithm can effectively train a robot in random motor failure scenarios and formulate a single robust policy for fault-tolerant robot control. Additionally, some methods [17], [20] initially employ model-free RL approaches to train expert policies tailored for each task. These expert strategies are then used to collect interaction data. Subsequently, offline RL algorithms, such as Decision Transformer [21], are applied to train the collected data across different tasks, with the objective of developing a versatile policy capable of handling a range of tasks. However, the aforementioned methods either pose challenges when deploying them to physical robot systems due to computational complexity or fail to ensure optimality under different joint stuck conditions. As a result, there is considerable scope for improvement in this field.

B. Meta Reinforcement Learning

The common state-of-the-art (SOTA) deep RL algorithms are mainly limited to single tasks. These algorithms require agents to interact extensively with the environment and then be tested in the same task. When testing on different tasks, the performance of these algorithms often deteriorates. Meta-RL presents a solution for multi-task learning by not only training the model to accomplish tasks but also enabling it to learn a "learning ability" that facilitates rapid adaptation to new tasks. Meta-RL methods can be classified into two types: gradient-based and context-based methods. Gradient-based Meta-RL methods aim to identify an appropriate set of neural network initialization parameters. When faced with a new task, after a few iterations using this parameter set, it is possible to achieve favorable results on the new task. The foundational work for this type of method is the Model Agnostic Meta-Learning (MAML) algorithm [22]. The First Order Meta-Learning (Reptile) algorithm [23] simplifies the training process based on MAML, while the Model Agnostic Exploration with Structured Noise (MAESN) algorithm [24] introduces structured noise to achieve structured exploration within the episode. The second category is context-based Meta-RL methods, which extract task-specific information from historical task samples and incorporate it into hidden variables. By adjusting its strategy based on these hidden variables, the learning algorithm can optimize its performance. For example, the Fast-RL via Slow-RL (RL^2) algorithm [25] employs a recurrent neural network (RNN) to adapt its hidden variables using historical interactive information, thus maximizing the cumulative rewards within each attempt. The Simple Neural Attentive Learner (SNAIL) algorithm [26] combines temporal convolutions (TC) and soft attention mechanisms to address the instability issues inherent in the RNN learning process. However, these methods are all based on on-policy RL methods, which have relatively low sampling efficiency. In contrast, we combine the setting of Meta-RL with off-policy RL methods to improve sampling efficiency and accelerate the training process.

III. Methodology

The objective of this work is to empower the robot to maintain a desired walking pattern, irrespective of any joint getting stuck. The complete framework is illustrated in Fig. 2. The subsequent sections provide a detailed description of the entire framework.

A. Problem Statement

The locomotion process of the quadruped robot can be represented as a Markov decision process (MDP), symbolized by the tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$. Where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ represents the action space, $p(s_{t+1}|s_t, a_t)$ signifies the state transition probability, $r(s_t, a_t)$ indicates the reward function and γ is the reward discount factor. The robot performs an action a_t at the current state s_t, receives a scalar reward r_t, and subsequently transitions to the next state s_{t+1} based on the transition probability.

As the locking variability of different joints modifies the robot's dynamic model, it leads to changes in state transition probability $p(s_{t+1}|s_t, a_t)$. This implies that for identical commands issued to the quadruped robot, varying joint stuck configurations will result in different walking patterns. Consequently, each joint stuck situation corresponds to a unique MDP, which we define as a task $\mathcal{T}$. We divide the set of all situations, denoted as $\Omega(\mathcal{T})$, into two parts: the training task set $\Omega_{train}(\mathcal{T})$ and the testing task set $\Omega_{test}(\mathcal{T})$. During the training stage, we sample tasks $\mathcal{T} \sim \Omega_{train}(\mathcal{T})$ to learn a policy. In the adaptation stage, when a specific task $\mathcal{T} \sim \Omega(\mathcal{T})$ is given, the agent collects a certain amount of data $D_{adapt}^{(\mathcal{T})}$ and adapts the trained policy to obtain the adapted policy $\pi_{\mathcal{T}}$. The objective of Meta-RL is to maximize the expected return of the adapted policy $\mathbb{E}_{\mathcal{T}\sim\Omega(\mathcal{T}),a_t\sim\pi_{\mathcal{T}}}[\sum_t \gamma^t r(s_t, a_t)]$.

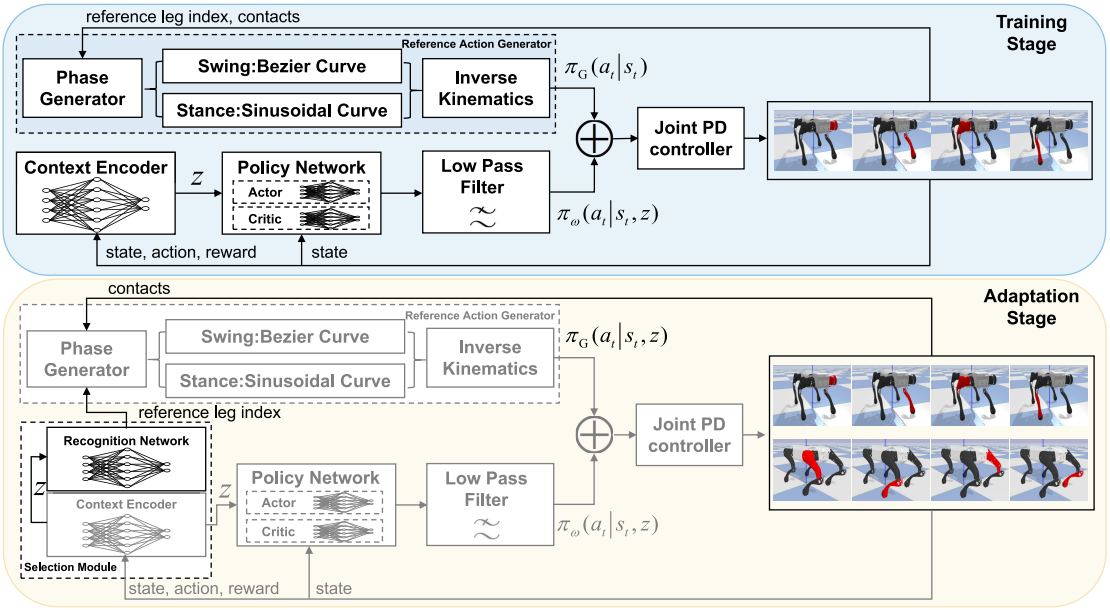


Fig. 2. Framework of our method. In the training stage, we select a reference leg for the RAG based on the stuck joint's index, training both the context encoder and policy network. The context encoder generates a latent vector z, which is derived from the contextual information and serves as input to the policy network for training. Additionally, we train a recognition network in a supervised manner. This network takes z as input and predicts the index of the leg where the stuck motor is located. During the adaptation stage, the robot, unaware of the specific stuck motor, executes a trajectory to generate z. This z helps the policy network produce $\pi_\omega(a_t|s_t, z)$, which is filtered to ensure smooth motion. Additionally, z is used by the recognition network to identify the faulty leg, guiding the selection of the reference leg for the RAG and obtaining $\pi_G(a_t|s_t, z)$. The robot's joint target positions are determined by integrating outputs from these two components, and joint torques are calculated via a PD controller for execution.

B. Design of Meta-RL

Utilizing contextual information to address Meta-RL tasks is an effective method. It leverages historical samples to infer task-related information, enabling the policy network to adjust its policy in a more targeted manner. Instead of designing policy networks that incorporate memory-based structures (*e.g.*, RNN, TC, *etc.*) to extract task-related information, we design the encoder network E_ϕ as a separate component from the policy network, as depicted in Fig. 2, where ϕ represents the parameters of the encoder. The historical information is referred to the context c. Specifically, $c_n^{\mathcal{T}} = (s_n, a_n, r_n, s_{n+1})$ represents an experience within task $\mathcal{T}$, and $c_{1:N}^{\mathcal{T}}$ encompasses all previous experiences. During the training phase, E_ϕ compresses the historical experience $c_{1:N}^{\mathcal{T}}$ into a distribution of context variables, facilitating the extraction of task-relevant information and its integration into latent vectors. As an MDP should be permutation invariant, *i.e.*, the order in which tuples are observed does not affect the inference of the task, we define the latent vector as follows:

$$z \sim \mathcal{N}(\Pi_{n=1}^{N} E_\phi^\mu(c_n^{\mathcal{T}}), \Pi_{n=1}^{N} E_\phi^\sigma(c_n^{\mathcal{T}})) \tag{1}$$

where $\mathcal{N}$ represents a normal distribution. E_ϕ^μ denotes the first half of the data outputted by E_ϕ, which serves as the mean of the normal distribution. E_ϕ^σ refers to the second half of the data outputted by E_ϕ, used to determine the variance of the normal distribution.

Inspired by the work presented in [27], we utilize the off-policy RL method soft Actor-Critic (SAC) [28] to integrate with the context encoder. Specifically, the first optimization objective of the encoder is designed as the critic's loss function, aimed at optimizing the state-action value function (Q-function). By associating z with the Q-function, the encoder is encouraged to extract information related to the Q-function estimation from the task set. The second component is to constrain the mutual information between z and c, which can be regarded as an information bottleneck method [29]. This helps to preserve task-relevant information and filter out task-irrelevant information, thereby avoiding model overfitting. This is achieved through the Kullback-Leibler (KL) divergence.

$$\mathcal{L}_{encoder} = \mathcal{L}_{critic} + \beta D_{KL}(\mathcal{N}(\Pi_{n=1}^{N} E_\phi^\mu(c_n), \Pi_{n=1}^{N} E_\phi^\sigma(c_n)) \| p(z)) \tag{2}$$

where β represents the weight coefficient and $p(z)$ is a unit Gaussian prior over z. $\mathcal{L}_{critic}$ denotes the critic's loss function, which aims to minimize the Bellman residual.

$$\mathcal{L}_{critic} = \mathbb{E}_{(s,a)\sim B, z\sim E_\phi}\left[\frac{1}{2}\left(\min_{i=1,2} Q_{\theta_i}(s,a,z) - \hat{Q}(s,a,z)\right)^2\right] \tag{3}$$

where θ denotes the network parameters of the critic (Q-network), i denotes various Q-networks, we utilize the smaller of the two estimated values, effectively preventing overestimation. The $\hat{Q}(s,z,a)$ can be calculated as follows.

$$\hat{Q}(s,a,z) = r + \gamma \mathbb{E}_{s'\sim B, z\sim E_\phi}\left[V_{\bar{\psi}}(s',z)\right] \tag{4}$$

where $V_{\bar{\psi}}$ represents the target value function, with its parameters set as an exponential moving average of the weights from the value function V_ψ. This method has been demonstrated to stabilize training. ψ is the value network's parameters. It is optimized by minimizing the squared residual error between

$V_\psi(s,z)$ and $V(s,z)$.

$$\begin{aligned}\mathcal{L}_{value} &= \mathbb{E}_{s\sim B,\bar{z}\sim E_\phi}\left[\frac{1}{2}\left(V_\psi(s,\bar{z}) - V(s,\bar{z})\right)^2\right] \\ &= \mathbb{E}_{s\sim B,\bar{z}\sim E_\phi}\Bigg[\frac{1}{2}\Big(V_\psi(s,\bar{z}) \\ &\quad -\mathbb{E}_{a\sim\pi}\Big[\min_{i=1,2} Q_{\theta_i}(s,a,\bar{z}) - \log\pi_\omega(a|s,z)\Big]\Big)^2\Bigg]\end{aligned} \tag{5}$$

where ω denotes the network parameters of the actor, and $\bar{z}$ signifies the exclusion of gradient compution. The actor can be improved by applying the following policy improvement formula.

$$\begin{aligned}&\mathcal{L}_{actor} \\ &= \mathbb{E}_{s\sim B,z\sim E_\phi}\left[D_{KL}\left(\pi_\omega(\cdot|s,z)\|\frac{\frac{1}{\alpha}\exp(\min_{i=1,2} Q_{\theta_i}(s,\bar{z},\cdot))}{\mathcal{Z}_\theta(s,z)}\right)\right]\end{aligned} \tag{6}$$

where α is the regularization coefficient, the partition function $\mathcal{Z}_\theta(s,z)$ is utilized to normalize the distribution, since it does not contribute to the gradient with respect to the new policy, we ignore it. Consequently, Eq. (6) can be further simplified as follows.

$$\mathcal{L}_{actor} = \mathbb{E}_{s\sim B,a\sim\pi,z\sim E_\phi}\left[\alpha\log\pi_\omega(a|s,z) - \min_{i=1,2} Q_{\theta_i}(s,a,\bar{z})\right] \tag{7}$$

It can also be interpreted as maximizing the value function $V(s,z)$.

C. Design of State / Action Space and Reward Functions

The state $s_t \in \mathbb{R}^{37}$ includes the base's linear velocity, roll, pitch, yaw, and angular velocity, as well as the angle and angular velocity of each joint. In addition, it contains four binary values indicating whether the foot is in contact with the ground.

The action in our method is the joint position residuals. Specifically, we incorporate the concept of residual learning to enhance the positive samples in the replay buffer and expedite the training process. The joint position command q_j^* sent to the robot consists of two components: the first component is the output of the Meta-RL, $a_t = \pi_\omega(a_t|s_t,z)$, and the second component is the output of the RAG, $a_t' = \pi_G(a_t|s_t)$, as explained in Section III-D. *i.e.*, $q_j^* = a_t + a_t'$. It is worth noting that the angle of the stuck motor remains constant regardless of the value of q_j^*. Furthermore, we employ a proportional-derivative (PD) controller to convert the joint angles into torques, as illustrated below:

$$\tau_j = k_p(q_j^* - q_j) + k_d(\dot{q}_j^* - \dot{q}_j) \tag{8}$$

where q_j^* and $\dot{q}_j^*$ denote the desired joint angles and angular velocities, respectively, with $\dot{q}_j^*$ set to zero. The variables q_j and $\dot{q}_j$ represent the current joint angles and angular velocities, respectively. Additionally, we manually specify the position and velocity stiffness using k_p and k_d, respectively.

TABLE I
REWARD FUNCTION DESIGN

Item	Equation	Weight
Base velocity	$r_1 = k_1 \times \exp\left(-5 \times \left\|\hat{v}_{x,y}^{base} - v_{x,y}^{base}\right\|\right)$ $k_1 = 1 - \tanh(\left\|w_1 \times (\hat{\Phi}_z - \Phi_z)\right\|^2)$	1.0
Base posture	$r_2 = 1 - \tanh(\left\|w_2 \times \sqrt{\Phi_x^2 + \Phi_y^2}\right\|^2)$	0.5
Feet velocity	$r_3 = k_3 \times \sum_{i=1}^{4} \min(\hat{v}_{x,y}^{feet,i}, v_{x,y}^{feet,i})/4$ $k_3 = 1 - \tanh(\left\|w_3 \times (\hat{\Phi}_z - \Phi_z)\right\|^2)$	0.2
Energy cost	$r_4 = \sum_{i=1}^{12} \lvert\tau_i \times \dot{q}_i\rvert \times t$	-0.1
Body contact	$r_5 = \sum_{i=1}^{N_l-4} TD_i^{others}$	-0.1
Feet contact	$r_6 = \max(\sum_{i=1}^{4}(1 - TD_i^{feet}) - 2, 0)$	-0.1

where $\hat{v}^{base}$ represents the target linearly velocity of the robot's body, while v^{base} represents the actual linearly velocity. $\hat{\Phi}$ denotes the desired euler angle of the body, whereas Φ represents the current euler angle. The target velocity of the foot is denoted as $\hat{v}^{feet}$, and the actual velocity is represented by v^{feet}. Furthermore, τ represents the torque of each joint, $\dot{q}$ signifies the speed of each joint, and t corresponds to the time required for each RL step. TD^{others} is a binary value indicating whether other parts of the robot (excluding the foot) are touching the ground, while TD^{feet} determines if the robot's foot is in contact with the ground. N_l denotes the total number of links in the robot. Additionally, w_1, w_2, and w_3 are predefined constants. The reward value r_t for each step is calculated as the weighted sum of the aforementioned factors.

Our reward function consists of six terms: r_1, which incentivizes the robot to track target velocity; r_2, which penalizes the rotation of the robot's body in the roll and pitch directions, it helps in maintaining the robot's balance; r_3, which encourages the movement of the robot's feet in the desired direction; r_4, which imposes a penalty on motor costs; r_5, which penalizes contact between the robot's body and the ground; and r_6, which incentivizes contact between the robot's feet and the ground. The detailed calculation formulas are provided in Tab. I.

D. Closed-Loop Reference Action Generator

1) Reference Action Generator: Drawing inspiration from the method outlined in [30], we devise RAG, which calculates the phase of each leg using the touchdown information of the reference leg l_{ref} and integrates it with predefined foot trajectories to yield the relative position of each foot to its corresponding abduction joint coordinate system at every moment. In contrast to [30], which consistently uses the front left (FL) leg as the reference leg, we incorporate a selection module to identify the injured leg, dynamically selecting an uninjured leg as l_{ref}. Furthermore, while the trajectory designed in [30] only allows the robot to move in a straight line forward, we enhance the foot trajectory to enable planar movement, significantly broadening the robot's mobility capabilities. Additionally, instead of using robotic dynamics models to solve a robot's joint torques as in [30], we employ inverse kinematics (IK) to determine the robot's joint angles. By introducing the RAG, we increase the number of positive

samples in Meta-RL, thereby greatly reducing the training difficulty.

Specifically, a motion cycle is defined as T_{stride}, which comprises two phases: the swing phase T_{sw} and the stance phase T_{st}. Specifically, $T_{stride} = T_{sw} + T_{st}$. The duration of the swing phase T_{sw} can be manually determined based on the specific situation. Besides, $T_{st} = \frac{2L_{span}}{v_d}$, where v_d represents the predetermined speed, and L_{span} is half the stride length.

The core of the RAG resides in the phase generation, which is designed based on the ground contact information of l_{ref}. During the training phase, the injured leg is known, allowing us to select an uninjured leg as l_{ref}. In the adaptation phase, with the injured leg unknown, we employ the Selection Module (which will be introduced in Section III-D2 subsequently) to detect the injured leg, similarly opting for an uninjured leg as l_{ref}. A touch-down event is defined when the z-direction force exceeds the threshold, indicating that l_{ref} has made contact with the ground. We denote this event with the Boolean value TD^{ref}, which is true when l_{ref} touches the ground and false during the swing phase. The phase of each leg can be calculated using the following formulas:

$$t_{ref}^{TD} = t \quad if \ TD^{ref} = true \tag{9}$$

$$t_{ref}^{elapse} = \begin{cases} t - t_{ref}^{TD} & if \ 0 < t_{ref}^{elapse} < T_{stride} \\ T_{stride} & if \ t_{ref}^{elapse} > T_{stride} \\ 0 & if \ t_{ref}^{elapse} < 0 \end{cases} \tag{10}$$

$$t_i = t_{ref}^{elapse} - \Delta S_{ref,i} T_{stride} \tag{11}$$

where t_{ref}^{TD} is the moment when l_{ref} touches down on the ground, t_{ref}^{elapse} is the elapsed time after t_{ref}^{TD}, and t_i represents the clock for each leg. $\Delta S_{ref,i}$ denotes the phase difference of leg i with respect to l_{ref}. We adopt the commonly used trot gait in this paper, wherein the phase of the FL leg aligns with that of the hind right (HR) leg, and the phase of the front right (FR) leg aligns with that of the hind left (HL) leg. Consequently, when selecting FL/HR as the reference leg, $\Delta S_{ref,i}$ is calculated using Eq. (12a). Similarly, when FR/HL is chosen as the reference leg, $\Delta S_{ref,i}$ is determined by Eq. (12b).

$$\Delta S_{trot}^{FL} = \Delta S_{trot}^{HR} = \begin{bmatrix} \Delta S_{ref,FL} \\ \Delta S_{ref,FR} \\ \Delta S_{ref,HL} \\ \Delta S_{ref,HR} \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.5 \\ 0.5 \\ 0.0 \end{bmatrix} \tag{12a}$$

$$\Delta S_{trot}^{FR} = \Delta S_{trot}^{HL} = \begin{bmatrix} \Delta S_{ref,FL} \\ \Delta S_{ref,FR} \\ \Delta S_{ref,HL} \\ \Delta S_{ref,HR} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.0 \\ 0.0 \\ 0.5 \end{bmatrix} \tag{12b}$$

We then normalize the clock of each leg by mapping t_i to $S_i(t)$. Specifically, when a leg is in the swing phase, $0 \leq S_i(t) \leq 1$, while in the stance phase, $1 \leq S_i(t) \leq 2$, see the formula below for details.

$$S_i(t) = \begin{cases} \dfrac{t_i + T_{stride}}{T_{st}} & -T_{stride} < t_i < -T_{sw} \\ \dfrac{t_i + T_{sw}}{T_{sw}} & -T_{sw} < t_i < 0 \\ \dfrac{t_i}{T_{st}} & 0 < t_i < T_{st} \\ \dfrac{t_i - T_{st}}{T_{sw}} & T_{st} < 0 < T_{stride} \end{cases} \tag{13}$$

After determining the phase, we utilize it to design the robot's foot trajectory. During the swing phase, a Bezier curve is employed as the reference trajectory. Conversely, during the stance phase, a sinusoidal curve is utilized.

$$p_i^{sw} = \begin{cases} p_{i,x}^{sw}(S_i(t)) = \sum_{k=0}^{n} c_k^{xy} B_k^n(S_i(t)) \cos\rho \\ p_{i,y}^{sw}(S_i(t)) = \sum_{k=0}^{n} c_k^{xy} B_k^n(S_i(t)) \sin\rho \\ p_{i,z}^{sw}(S_i(t)) = \sum_{k=0}^{n} c_k^{z} B_k^n(S_i(t)) \end{cases}$$

$$s.t. \ 0 \leq S_i(t) \leq 1 \tag{14}$$

$$p_i^{st} = \begin{cases} p_{i,x}^{st}(S_i(t)) = L_{span}(1 - 2S_i(t)) \cos\rho \\ p_{i,y}^{st}(S_i(t)) = L_{span}(1 - 2S_i(t)) \sin\rho \\ p_{i,z}^{st}(S_i(t)) = \sigma \cos\left\{\dfrac{\pi}{2L_{span}}\left[p_{i,x}^{st}(S_i(t)) + p_{i,y}^{st}(S_i(t))\right]\right\} \end{cases}$$

$$s.t. \ 1 \leq S_i(t) \leq 2 \tag{15}$$

where p_i^{sw} denotes the position of the robot's foot relative to the abduction joint coordinate system during the swing phase, with $p_{i,x}^{sw}$, $p_{i,y}^{sw}$, and $p_{i,z}^{sw}$ representing its coordinates in the three directions, respectively. c_k^{xy} refers to the k-th control point on the xy plane, c_k^z represents the k-th control point in the z direction, and $B_k^n(S_i(t))$ denotes the Bernstein polynomial of degree n. Additionally, ρ denotes the trajectory's rotation angle relative to the robot's forward direction. On the other hand, p_i^{st} refers to the foot's position during the stance phase, with σ signifying the amplitude variable. By utilizing IK, the robot's reference joint position can be determined based on these variables.

$$\pi_G(a_t|s_t) = IK(p_i^{sw}, p_i^{st}) \tag{16}$$

where s_t refers to the foot contact information of the robot, which is utilized to determine whether a touch-down event has taken place.

2) Selection Module: During the adaptation stage, the index of the robot's stuck motor is often unavailable. When the joint in the reference leg becomes stuck, there is a risk that it may fail to touch down on the ground. This may lead to disordered gait in robots, consequently degrading the performance of the RAG. To tackle this challenge, we develop a selection module that comprises two components: the context encoder, which has been previously trained, and the recognition network. The recognition network takes z as input, derived from historical information $c_{1:N}^{\mathcal{T}}$ using the context encoder. It outputs the probability of each leg being injured. We train the recognition

Algorithm 1 Training Stage

Initialize replay buffer B_i and C_i for each task $\mathcal{T}_i$.
for *epoch* in pre-defined steps **do**
  for each $\mathcal{T}_i$ **do**
    Initialize and empty context replay buffer C_i;
    Specify the reference leg l_{ref} according to $\mathcal{T}_i$;
    for $t = 1, \ldots, T$ **do**
      Sample $c \sim C_i$, get z according to Eq. (1);
      Get $a'_t = \pi_G(a_t|s_t)$ according to Eq. (16);
      Get $a_t = \pi_\omega(a_t|s_t, z)$ and filter it;
      Execute the desired joint angles $q^*_j = a_t + a'_t$;
      Obtain reward r_t and transfer to next state s_{t+1};
      Store $\langle s_t, a_t, r_t, s_{t+1}\rangle$ to B_i and C_i;
    end for
  end for
  for *step* in training steps **do**
    for each $\mathcal{T}_i$ **do**
      Sample context $c_i \sim C_i$ and RL batch $b_i \sim B_i$;
      Get z according to Eq. (1);
      Calculate $\mathcal{L}^i_{encoder}$ according to Eq. (2);
      Calculate $\mathcal{L}^i_{value}$ and $\mathcal{L}^i_{critic}$ according to Eq. (5) and Eq. (3), respectively;
      Calculate $\mathcal{L}^i_{actor}$ according to Eq.(7);
    end for
    Update parameters of all networks;
  end for
end for

Algorithm 2 Adaptation Stage

Initialize context $c^{\mathcal{T}} = \{\}$.
Specify reference leg as $l_{ref} = FL$.
for *step* in predetermined steps **do**
  Sample $z \sim \mathcal{N}(0, 1)$;
  Get $\langle s_t, a_t, s_{t+1}, r_t\rangle$ according to line 8-11 in Algorithm 1;
  Accumulate context $c^{\mathcal{T}} = c^{\mathcal{T}} \cup \langle s_t, a_t, s_{t+1}, r_t\rangle$;
end for
Calculate z according to Eq. (1);
Get injured leg by selection module and re-choice l_{ref};
for *step* in predetermined steps **do**
  Get $a'_t = \pi_G(a_t|s_t, z)$ under updated l_{ref};
  Get $a_t = \pi_\omega(a_t|s_t, z)$ under updated z;
  Execute the desired joint angles $q^*_j = a_t + a'_t$;
  Obtain reward r_t and transfer to next state s_{t+1};
end for

network in a supervised manner, employing the cross-entropy loss as the loss function.

$$\mathcal{L}_{recog} = -\frac{1}{N_s}\sum_{i=1}^{N_s}\sum_{j=1}^{4}\hat{y}_{ij}\log(y_{ij}) \tag{17}$$

where N_s is the number of samples, y_{ij} represents the predicted probability that the i-th sample belongs to the j-th category, and $\hat{y}_{ij}$ represents the true label. In the adaptation stage, the FL leg can be used as the reference leg to move a short distance, after which the current stuck motor can be identified by the collected tuples, allowing for the adaptive selection of the reference leg for the RAG. In summary, the pseudocode for the training phase is presented as Algorithm 1, while the adaptation phase is outlined in Algorithm 2.

IV. Experiment

A. Experimental Setup

1) Training Details: During training, we utilize Pybullet as the simulator, with an update frequency set to 1000 Hz. The algorithm's actions are repeated 20 times, resulting in an algorithm frequency of 50 Hz. The URDF model of the robot is based on the Jueying Lite3 from DeepRobotics company. If the robot's body contacts the ground or the posture of the body exceeds a certain threshold, the current epoch will be terminated prematurely. Tab. II presents the values of hyperparameters used in the algorithm. Moreover, to enhance policy robustness, we employ an asymmetric noise strategy [31], as illustrated in Tab. III, where only the actor's inputs are perturbed with noise while the critic's inputs remain noise-free. This method allows the robot to be resilient to noise without compromising the algorithm's performance.

TABLE II
Hyperparameter Settings

Hyperparameters	Values
Optimizer	Adam
Actor/Critic/Context Encoder/Recognition Network Learning rate	3×10^{-4}
Discount factor	0.99
Replay buffer size	1×10^{6}
Target smoothing coefficient	5×10^{-3}
KL lambda	1
Samples per mini-batch (Actor/Critic)	256
Samples per mini-batch (Context Encodr/Recognition Network)	64
Nonlinearity	ReLU
Context Encoder/Recognition Network architecture	[200,200,200]
Actor/Critic architecture	[300,300,300]

TABLE III
Range of State Noise

State	Sample type	Range
Linear velocity	Normal	(0, 0.1) + $value_{ori}$
Base orientation	Normal	(0, 0.1) + $value_{ori}$
Base angular velocity	Normal	(0, 0.5) + $value_{ori}$
Motor position	Normal	(0, 0.01) + $value_{ori}$
Motor velocity	Normal	(0, 1.0) + $value_{ori}$

2) Motor Stuck Setting: We record the range of each joint during normal movement. Based on this range, we employ a curriculum training [32] method. Specifically, we establish five curriculum levels. During the initial 1/5 phase of the training process, we sample angles of stuck joints within the range of level 0. In the phase from 1/5 to 2/5, we conduct sampling within the range of level 1, and so on. The specific values are outlined in Tab. IV. The joint angles associated with level 0 correspond to the angles at which the robot maintains a standing position. In particular, we evaluate the algorithm's performance using forward locomotion experiments and diagonal locomotion experiments. As shown in

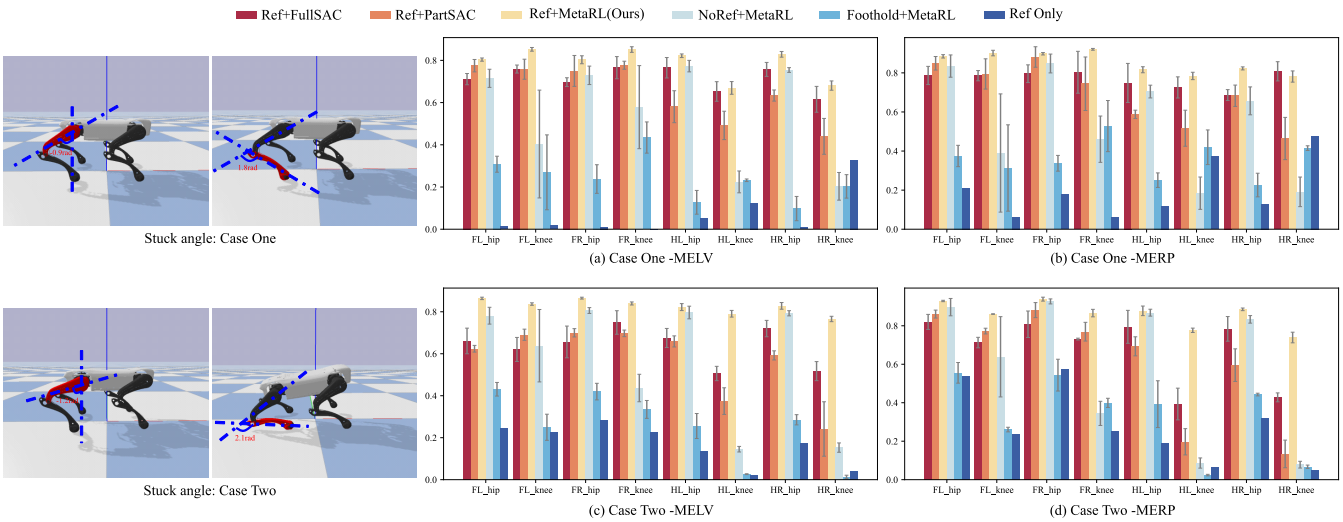


Fig. 3. Comparison of various ablation methods and the proposed method under different motor stuck cases. Except for **Ref-Only**, all results are derived from three different random seeds. The x-axis of the bar chart represents the different stuck joints, while the y-axis corresponds to the values of MELV or MERP.

TABLE IV
CURRICULUM SAMPLING RANGE

Joint	Angle Range During Movement	Curriculum Level	Sample Range
Abduction	Forward: (-0.04, 0.04) Diagonal: (-0.26, 0.14)	0	0
		1	(-0.1, 0.1)
		2	(-0.15, 0.15)
		3	(-0.2, 0.2)
		4	(-0.3, 0.2)
Hip	Forward/Diagonal: (-1.35, -0.66)	0	-0.9
		1	(-1.0, -0.8)
		2	(-1.1, -0.7)
		3	(-1.2, -0.6)
		4	(-1.4, -0.6)
Knee	Forward/Diagonal: (1.77, 2.35)	0	1.8
		1	(1.7, 1.9)
		2	(1.7, 2.0)
		3	(1.7, 2.2)
		4	(1.7, 2.4)

Tab. IV, in the forward locomotion experiment, the swing range of the abduction joint is extremely small, indicating that even if the abduction joint is locked, it has minimal impact on the robot's locomotion performance. Therefore, for this experiment, we only consider the cases where the hip and knee joints are locked (a total of eight tasks). In the diagonal locomotion experiment, the swing range of the abduction joint is larger and can significantly affect the robot's locomotion performance. Hence, in this task, we consider all cases where any joint is locked (a total of twelve tasks).

3) Evaluation Metrics: Inspired from [33], our analyses are based on two metrics: Mean Episode Linear Velocity Tracking Reward (MELV) -the average of r_1 per episode as outlined in Tab. I, which gauges the robot's proficiency in tracking target velocity. A higher MELV indicates superior performance. The second metric is the Mean Episode Roll Pitch Tracking Reward (MERP) -the average of r_2 per episode as shown in Tab. I, assessing the stability of the robot's posture, where a greater MERP signifies better stability.

B. Ablation Study

1) Ablation Study of Meta-RL: In this section, we conduct four ablation experiments to evaluate the influence of Meta-RL on the forward motion task. The experimental setup for each experiment is outlined below.

Ref+FullSAC: In this set of experiments, we use RAG to obtain π_G, while π_ω is obtained through the SAC algorithm. Typically, RL algorithms are only applicable to individual tasks. In order to make the SAC algorithm applicable to various tasks, we adopt the domain randomization method. Specifically, during the robot training process, the stuck motors are randomly sampled from all tasks, *i.e.,* {*FL_hip*, *FL_knee*, *FR_hip*, *FR_knee*, *HL_hip*, *HL_knee*, *HR_hip*, *HR_knee*}, allowing the algorithm to encounter each task during training.

Ref+PartSAC: As with Ref + FullSAC, π_G is derived using RAG, and π_ω via SAC. Unlike Ref + FullSAC, during training, stuck motors are randomly sampled exclusively from training task set {*FL_hip*, *FL_knee*, *FR_hip*, *FR_knee*}. This method allows us to assess the generalization capabilities of the conventional RL algorithm on tasks it has not previously encountered, *i.e.*, the testing task set {*HL_hip*, *HL_knee*, *HR_hip*, *HR_knee*}.

Ref+MetaRL: The proposed method. π_G is obtained via RAG, while π_ω is derived through Meta-RL. The training and testing task set follows that of Ref + PartSAC, where only tasks from the training set are encountered during the training process. By comparing it with Ref + PartSAC, we can validate the advantages of Meta-RL over conventional RL methods in enhancing generalization capabilities.

Ref-Only: Utilizing only the RAG to facilitate robot motion, serves as the lower bound for various ablation algorithms.

We conduct tests on the trained algorithms and show their results in Fig. 3. To validate the algorithm's efficiency across a variety of stuck angles, we select two distinct stuck angles for each joint, as illustrated on the left side of Fig. 3. Furthermore, we showcase the accumulative rewards in Fig. 4. Based on the results presented in Fig. 4, the proposed method achieves the highest rewards, followed by Ref + PartSAC and

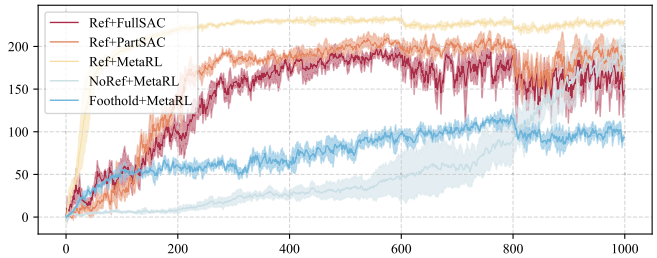


Fig. 4. The accumulative rewards of ablation study. The x-axis represents the number of training epochs. These results are averaged across three different seeds, with the shaded areas indicating the variance among these seeds.

Ref + FullSAC. Additionally, Fig. 3 indicates that the proposed method has almost achieved the highest value in both criteria, for both the training task set and the testing task set, with no significant difference between the two sets (within each subplot, the first four groups are allocated to the training set, whereas the latter four groups are reserved for the testing set). Under normal conditions (where no motor stuck occurs), Ref-Only enables the robot to move forward. However, in the event that the motor is stuck, Ref-Only fails to maintain the robot's expected motion state, leading to flipping or severe trembling. For Ref + PartSAC, while it achieves commendable results in the training set, its performance significantly deteriorates in the testing set. These findings demonstrate the weak generalization of conventional RL algorithms (*i.e.*, Ref + PartSAC) on unseen tasks and highlight the effectiveness of the Meta-RL settings.

Unlike Ref + PartSAC, Ref + FullSAC has achieved favorable outcomes on both the training and testing tasks, a result of its exposure to all tasks during the training process. It is worth noting that, the proposed method, which has not been trained on the testing tasks, surpasses Ref + FullSAC in all tasks across training and testing sets. This can be attributed to Ref + FullSAC's compromise of task-specific optimality in pursuit of general applicability across multiple tasks. These results suggest that the Meta-RL setting can provide better policies for various tasks compared to domain randomization methods (*i.e.*, Ref + FullSAC).

2) Ablation Study of RAG: In this section, we conduct three sets of experiments to demonstrate the necessity of residual design (*i.e.*, whether using a RAG), The experimental design for the **Ref+MetaRL** method aligns with the previous section, while the settings for the other two methods are as follows.

NoRef+MetaRL: Without utilizing the RAG, the robot's initial default joint angles are set to $q^*_{init} = ([0, -0.9, 1.8] \times 4)$ rad, and the final angles of each joint are calculated as $q^*_j = q^*_{init} + a_t$. The action a_t is selected by the Meta-RL algorithm in the proposed method. The design of the training and testing sets is identical to that of Ref + PartSAC/Ref + MetaRL.

Foothold+MetaRL: In contrast to our proposed method, the correction is carried out in Cartesian space to adjust the foothold position, which is similar to the method presented in [34]. Specifically, the reference foothold point is obtained using the RAG, and then the Meta-RL algorithm corrects its position. The final joint positions are determined using leg IK: $q^*_j = IK(p_{ref} + a_t)$. Where p_{ref} represent the reference foothold positions obtained from the RAG (as shown in Eq. (14) and Eq. (15)), while a_t represent the corrected foothold positions obtained from the Meta-RL algorithm. The design of the training and testing sets is consistent with Ref + PartSAC/Ref + MetaRL.

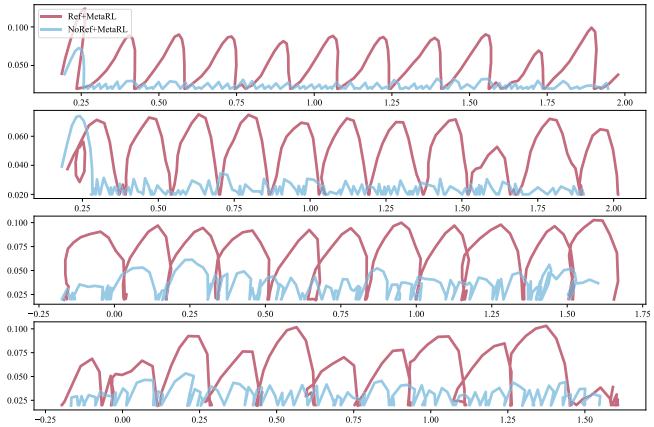


Fig. 5. Trajectories of the robot's foot under Ref+MetaRL and NoRef + MetaRL methods. From top to bottom, the order is FL, FR, HL, and HR. The x-axis represents the robot's forward direction, while the y-axis indicates the height of the robot's foot from the ground.

Fig. 3 reveals that the NoRef + MetaRL method exhibits poor performance in various knee-stuck tasks. In the absence of RAG guidance, the control strategies learned tend to drive the robot forward by shuffling, as shown in Fig.5, leading to excessive wear on the robot's feet and challenges in deploying to physical robots. In our proposed method, incorporating a RAG allows for rapid accumulation of positive samples in the replay buffer, thereby mitigating sparse rewards and enhancing the algorithm's performance. Additionally, this method ensures locomotion through stepping rather than shuffling.

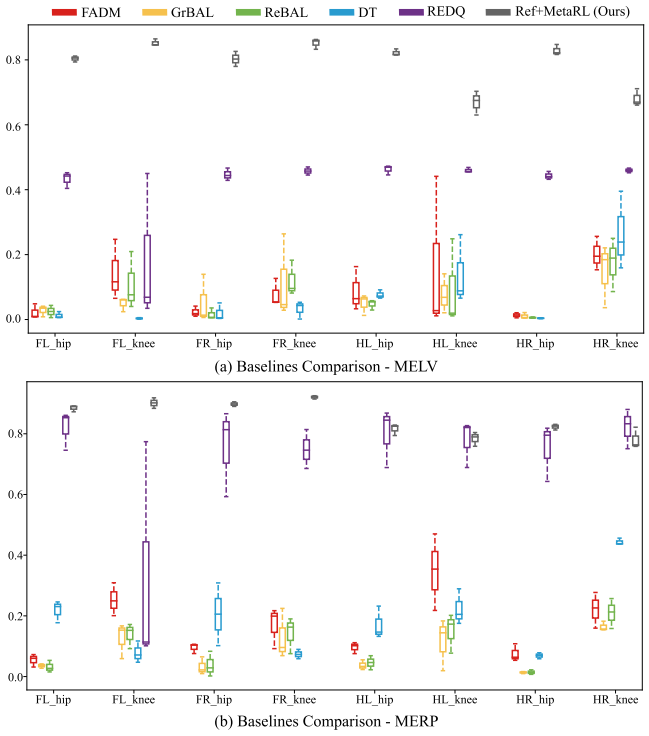
Fig. 3 demonstrates that among all ablation studies, except for the Ref-Only method serving as the lower bound, Foothold + MetaRL exhibits the poorest performance, occasionally even underperforming Ref-Only. While previous work [34] successfully employs RL to correct foothold positions on uneven ground, it fails to perform well in our task. Since the joints of each leg are coupled, IK becomes problematic when the joints are stuck. Therefore, our task benefits more from direct optimization in joint space rather than optimizing the foothold position in the Cartesian space.

C. Comparison With Baselines

In this section, we compare the proposed method with three model-based RL methods and two model-free RL methods, which are listed as follows.

Fast Adaptation Dynamics Model (FADM, [12]): Model-based RL method, it utilizes an ensemble-based dynamics model that integrates future trajectory and damage information to learn the dynamics model. The actions are selected via CEM, which are based on the predicted states.

Gradient-Based Adaptive Learner (GrBAL, [11]): Model-based RL method, it leverages gradient-based methods to enable online adaptation, specifically by utilizing MAML [22] to optimize an adaptation meta-objective for training a dynamics model.



(a) Baselines Comparison - MELV

(b) Baselines Comparison - MERP

Fig. 6. The results of comparison experiments. The x-axis represents the motor indexes, and the y-axis represents the criteria.

Recurrence-Based Adaptive Learner (ReBAL, [11]): Model-based RL method, which shares similarities with GrBAL. It employs RNN to learn its own update rule via an internal gating structure.

Decision Transformer (DT, [17]): Model-free RL method, this method initially focuses on training separate control policies for various joint stuck tasks to gather interaction data. These data are then used to train a network based on the Decision Transformer, ultimately yielding a control policy with generalization capabilities.

Randomized Ensembled Double Q-Learning (REDQ, [35]): A SOTA model-free RL algorithm, which is based on the SAC algorithm, utilizes the mean of multiple critics for the value function, which significantly reduces the variance in value estimation and greatly enhances algorithm performance. Furthermore, we employ domain randomization techniques to expose the method to all joint stuck tasks during training, enhancing its generality.

Fig. 6 presents the comparison results of various baselines using the same evaluation metrics as the ablation experiments. For the MELV metric, FADM, GrBAL, ReBAL, and DT methods perform relatively poorly, while REDQ performs better, but its values are only about half of those of the proposed method. Additionally, for the *FL_knee* joint, REDQ exhibits a wider range of data distribution, indicating sensitivity to random seed selection, whereas the proposed method shows a narrower distribution, demonstrating robustness to random seed choice. Regarding the MERP metric, the three model-based methods and DT still yield unsatisfactory results. For REDQ, the data distribution range remains large, especially for the *FL_knee* joint. In the cases of *HL_hip*, *HL_knee*, and *HR_knee*, REDQ's median exceeds that of the proposed method because of domain randomization, which is trained for testing tasks but not for the proposed method. Nevertheless, considering both metrics holistically, the proposed method still outperforms REDQ.

TABLE V
INFERENCE TIME COMPARISON

Methods	FADM	GrBAL	ReBAL	DT	REDQ	Ours
Times	0.0875 ± 0.1050	0.1968 ± 0.0341	0.1804 ± 0.0199	0.0010 ± 0.0002	0.0008 ± 0.0008	0.0006 ± 0.0007

We suspect that the poor performance of the model-based methods may be attributed to the high degree of freedom in the quadruped robot, leading to a complex dynamics model and increased difficulty in model learning. Modeling errors can also have a negative impact on the policies. As for the DT method, we attribute its performance to the use of collected offline data for training. The limited size and coverage of the offline dataset prevent it from effectively covering the state and action spaces, hindering the achievement of the policy comparable to that of learning through interaction with the environment.

Additionally, we have calculated the inference time for each step of different algorithms, as presented in Tab. V. It can be observed that the three model-based methods are comparatively time-consuming due to the numerous iterations required in the planning process of CEM/RS. In contrast, our method and two model-free methods provide rapid inference times, making them suitable for deployment on quadruped robots that demand high control frequency.

D. Analysis of Selection Module

We collect 100 sets of latent vectors and their corresponding indices for each stuck joint, apply t-distributed Stochastic Neighbor Embedding (t-SNE) to reduce the dimensionality, and plot the results in Fig. 10 (a). It can be observed that the t-SNE method effectively distinguishes the hip and knee joints of each leg. However, the latent vectors corresponding to the abduction joints of the four legs are mixed, making it difficult to differentiate. This aligns with our previous analysis, which suggests that the impact of the abduction joints on the robot's motion is smaller, as they have a smaller swing amplitude compared to the hip and knee joints.

The selection module is capable of obtaining the injured leg index, which aids in the selection of a suitable reference leg for the RAG. To evaluate the effectiveness of the selection module, we classify the aforementioned data using the trained recognition network and output the injured leg index corresponding to the latent vectors. The confusion matrix in Fig. 10 (b) shows that out of 1200 data points, only one is misclassified, resulting in an accuracy of 99.9%. Compared to the dimensionality reduction method t-SNE, the selection module demonstrates better prediction of the injured leg index, providing evidence of its effective identification of injured legs.

E. Multiple Scenes Experiments

1) Diagonal Motion Experiment: In the previous experiments, the objective is to drive the robot forward along the

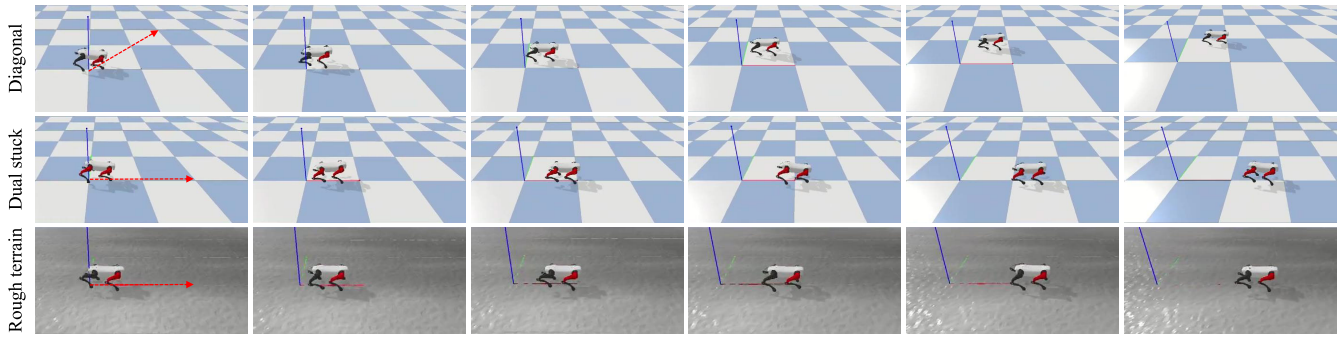


Fig. 7. Snapshots of the proposed method under multiple-scene experiments in the simulator. From top to bottom, they are the diagonal motion experiment, the dual-joint stuck experiment, and the rough terrain experiment. The link highlighted in red corresponds to the stuck joint, and in this case, we have chosen the *FL_hip* joint as an example.

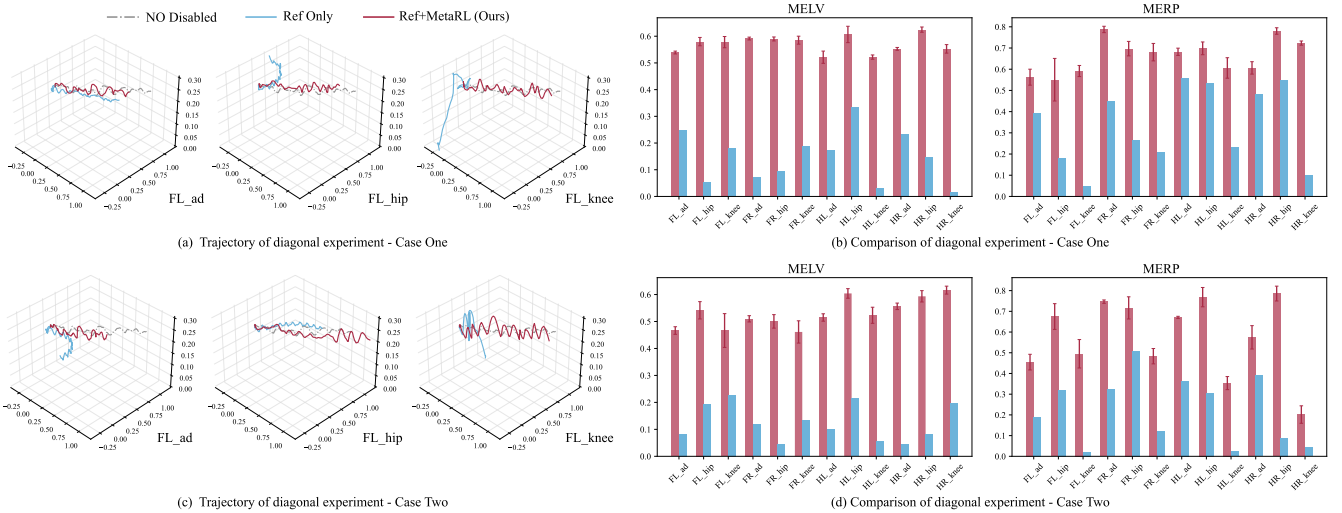


Fig. 8. (a) and (c). The trajectory of the robot in the diagonal motion experiment. In Case One, the abduction joint is set to 0.0 rad, the hip is set to -0.9 rad, and the knee is set to 1.8 rad. In Case Two, the abduction joint is set to 0.1 rad, the hip is set to -1.2 rad, and the knee is set to 2.1 rad. Due to space limitations, we only show the trajectory under the FL leg. Each coordinate axis represents the position of the robot's COM in space. The robot starts at coordinates (0, 0, 0.26) and executes a diagonal movement towards the front left. (b) and (d). Comparison of the proposed method and the Ref-Only method in terms of MELV and MERP metrics.

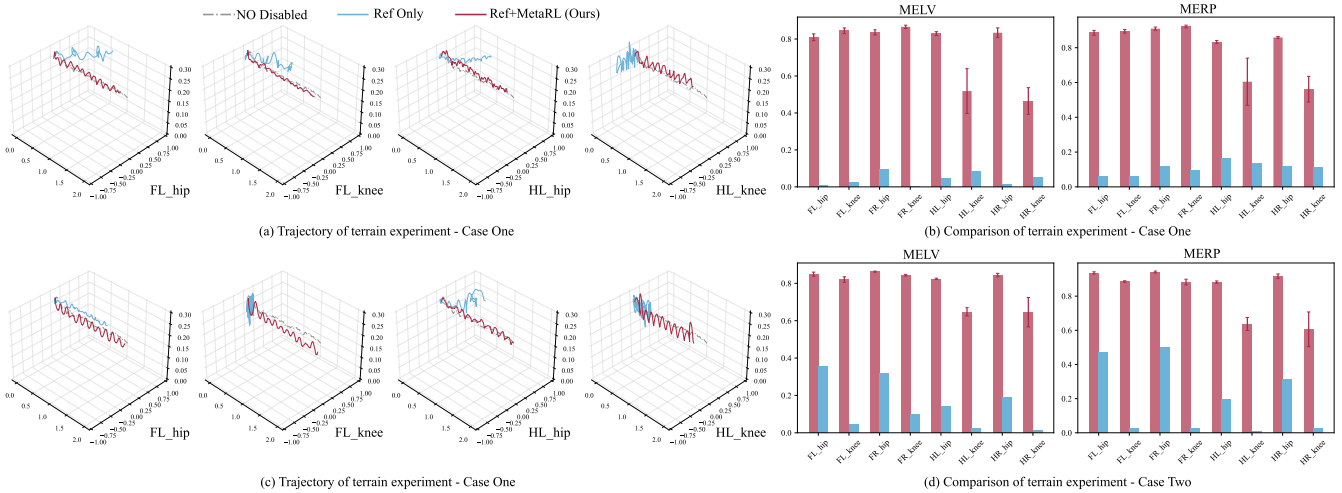
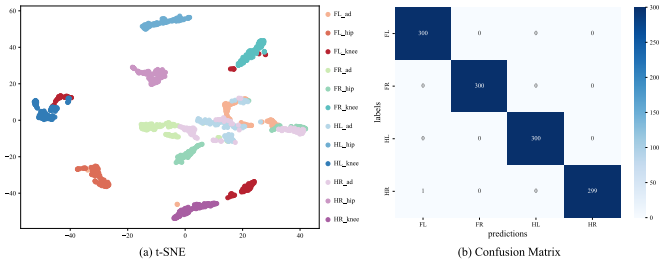


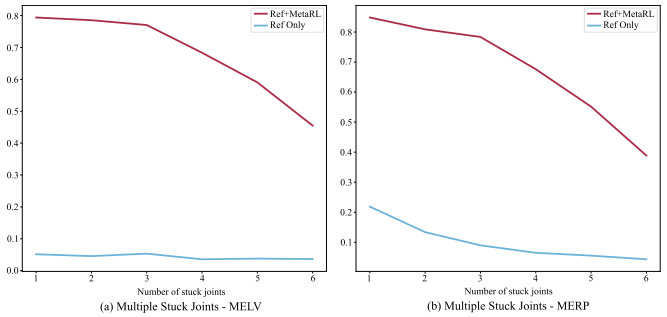
Fig. 9. (a) and (c). The trajectory of the robot in the rough terrain experiment. The design of Case One and Case Two aligns with Fig. 3. Due to space limitations, we only show the trajectory under the FL leg and HL leg. Each axis denotes the position of the robot's COM in space. The robot initiates at coordinates (0, 0, 0.26) and performs a forward movement along the x-axis. (b) and (d). A comparative analysis of the proposed method and the Ref-Only method based on MELV and MERP metrics.

x-axis, indicating that the rotation angle ρ in Eq. (14) and Eq. (15) is set to zero. To evaluate the robot's omnidirectional motion capabilities with a stuck joint, we conduct an experiment in this section to make the robot follow a diagonal trajectory while maintaining a desired yaw angle. Unlike walking straight along the x-axis, the trajectory is affected by the abduction joints. Hence, in this section, we verify the occurrence of any of the twelve joints getting stuck.



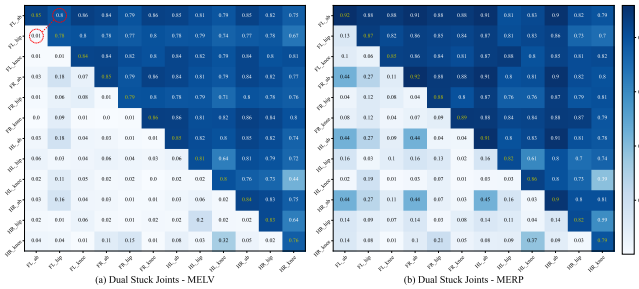
(a) t-SNE (b) Confusion Matrix

Fig. 10. (a). t-SNE visualization for the latent vector of context encoder on different joints. (b). Confusion matrix of the selection module, each element in the matrix represents the number of model predictions where the predicted category is along the horizontal axis and the actual category is along the vertical axis.



(a) Multiple Stuck Joints - MELV (b) Multiple Stuck Joints - MERP

Fig. 11. The relationship between the performance of the two methods and the number of stuck joints. The x-axis denotes the number of stuck joints, while the y-axis illustrates the performance.



(a) Dual Stuck Joints - MELV (b) Dual Stuck Joints - MERP

Fig. 12. Heatmap of the dual-joint stuck experiment, each grid represents the value of metric, with the horizontal and vertical axes representing the specific stuck joints. The upper right area corresponds to the results of Ref+MetaRL, while the lower left area corresponds to the results of Ref-Only. The yellow numbers on the diagonal represent single-joint stuck and can be disregarded.

Fig. 8 (a) and (c) depict the COM's trajectory of the diagonal motion experiment. Upon analyzing the trajectory labeled as Ref-Only, we observe a significant impact on the robot's trajectory due to motor locking, where the robot even fell due to *FL_knee* locking. By implementing our proposed method, the robot's trajectory closely aligns with the target trajectory. Fig. 8 (b) and (d) demonstrate the comparison of the two methods in terms of MELV and MERP metrics. It can be observed that the proposed method significantly improves the values of these metrics in all tasks. These experiments demonstrate the effectiveness of our proposed method in achieving omnidirectional locomotion in quadruped robots. Previous research on fault-tolerant strategies [11], [12], [14], [16] for quadruped robots lack experimentation involving the robot's omnidirectional movement.

2) Multiple Joints Stuck Experiment: The quadruped robot has six degrees of freedom and twelve degrees of actuator, providing redundancy. Therefore, theoretically, to ensure normal robot motion, the number of stuck joints should be at most six. We train corresponding models for scenarios involving stuck joints from one to six and record the performance of the proposed method and Ref-Only method across two metrics, MELV and MERP, as shown in Fig. 11. It can be observed that the proposed method significantly outperforms the Ref-Only method across various numbers of stuck joints. However, as the number of stuck joints increases, the robot's performance on both metrics sharply declines, aligning with our intuition that more stuck joints make it increasingly difficult for the robot to maintain the desired motion state.

Furthermore, we validate the motion performance of the proposed method during dual-joint stuck conditions. Fig. 12 illustrates the results, evaluated using the MELV and MERP metrics. The bottom left corner represents the results of Ref-Only, while the top right corner represents the results of Ref +[]MetaRL. For instance, in (a), the values of 0.01 and 0.8, highlighted by red circles, correspond to the results of Ref-Only and Ref + MetaRL methods, respectively, when *FL_ab* and *FL_hip* joints are locked. Through Fig. 12 (a) and (b), it is evident that the proposed method significantly enhances the robot's motion performance during dual joints stuck. For a more intuitive comparison, Fig. 7 displays the results of the proposed method when *FR_hip* and *HR_hip* joints are locked.

3) Rough Terrain Experiment: To validate the robustness of the proposed method across different terrains, we modify the ground in the simulator to rough terrain as depicted in the third row of Fig. 7. Fig. 9 (a) and (c) illustrate the trajectories of robots on the rough terrain, while (b) and (d) present the comparative results between the proposed method and Ref-Only based on two aforementioned metrics. Analyzing Fig. 9 (a) and (c), it can be observed that adopting the Ref-Only method results in severe oscillations in the robot. By employing the proposed method, the robot can better track the target trajectory (*i.e.*, No-Disabled). Furthermore, analyzing Fig. 9 (b) and (d), it can be noted that the adoption of the proposed method significantly improves both MELV and MERP values for all joint stuck scenarios.

F. Real-World Experiment

1) Deployment Details: We conduct real-world experiments using the Jueying Lite3 robot. It weighs 22 kg and is powered by brushless electric motors that can reach a maximum speed of 10.6 rad/s and a maximum torque of 18 Nm. The robot is equipped with one Inertial Measurement Unit (IMU) and 12 motor encoders, which gather essential data about the robot's state, including body orientation, body angular velocity, joint positions, and joint velocities. The ground contact forces are detected using current loops. In the physical environment, a common method of estimating the velocity of a quadruped robot involves the use of an Extended Kalman Filter (EKF) [36], which integrates data from IMU, motor encoders, and foot-contact forces to estimate the robot's velocity. However, due to the joint stuck in our case, the foot-contact force information becomes inaccurate, leading to the failure of velocity estimation. To address this issue, we have employed the Intel Real Sense T265 camera, which is equipped with

Fig. 13. Deployment results on the physical robot, the angles at which the joints become stuck are consistent with Case One in Fig. 3. Due to limited spaces, *FL_hip* and *FL_knee* are chosen to represent the training set, while *HR_hip* and *HR_knee* are selected for the testing set. For each joint, we display the results obtained using two methods: Ref-Only and Ref+MetaRL. The side view is located in the top-right/top-left corner of the front view. The target direction is indicated by the red arrows, while the red circles highlight the motors that are stuck. Due to the placement of the camera on the robot's left side when taking side views of *FL_hip* and *FL_knee*, and on the right side for *HR_hip* and *HR_knee*, the target directions depicted in these side views are opposite.

stereo fisheye cameras and an IMU. It uses a built-in Visual-Inertial Odometry (VIO) algorithm to fuse data from multiple sensors and obtain velocity information of the camera. Since the camera is fixed to the robot via a 3D-printed connector, its velocity is representative of the robot's velocity. During deployment, we utilize the API provided by the Pyrealsense2 library[1] to acquire the robot's velocity at a frequency of 50 Hz, which is then used as an input for the Meta-RL policy.

The trained networks are converted to the ONNX format and directly deployed on the robot without fine-tuning. To prevent excessive changes to joint positions and enhance the motor's ability to track target instructions, we apply a low-pass Butterworth filter to smooth the instructions obtained from the Meta-RL algorithm. The Meta-RL inference frequency is set to 50 Hz, and we achieve an actual command frequency of 1000 Hz issued to the robot by linearly interpolating between two consecutive target joint commands. This frequency is consistent with the simulator. Besides, we utilize pybind11[2] to convert the C ++ interface provided by DeepRobotics company into a Python interface, enabling access to algorithm inputs and outputs via the Python interface.

2) Results: Fig. 13 and Fig. 14 illustrate the motion performance of the robot under different joint stuck angles, comparing the Ref-Only and Ref + MetaRL methods. From both figures, it is evident that utilizing the Ref-Only method alone may result in the robot deviating from the intended direction or even experiencing severe body shaking, leading to uncontrollable results. By employing the Ref + MetaRL method, the quadruped robot's inherent redundancy allows for fine-tuning of other joint angles through Meta-RL outputs, enabling the robot to maintain its original direction and minimize body shaking. This ensures that the robot can reach the designated position despite the joint lock. Moreover, analysis of results from both the training and testing sets reveals that our method maintains efficacy in new tasks (as represented by the HR leg), demonstrating the

[1]https://pypi.org/project/pyrealsense2/

[2]https://github.com/pybind/pybind11

Fig. 14. Deployment results on the physical robot, the angles at which the joints become stuck are consistent with Case Two in Fig. 3. Due to limited spaces, *FL_hip* and *FL_knee* are chosen to represent the training set, while *HR_hip* and *HR_knee* are selected for the testing set. For each joint, we display the results obtained using two methods: Ref-Only and Ref+MetaRL. The side view is located in the top-left corner of the front view. The target direction is indicated by the red arrows, while the red circles highlight the motors that are stuck. Due to the placement of the camera on the robot's left side when taking side views of *FL_hip* and *FL_knee*, and on the right side for *HR_hip* and *HR_knee*, the target directions depicted in these side views are opposite.

effectiveness of the developed fault-tolerant strategies under physical environments.

V. Conclusion

In this paper, we present a novel fault-tolerant control strategy for quadruped robots to address the issue of joints getting stuck. Addressing the limitations of previous research, we leverage the Meta-RL method to model joint locking in quadruped robots for the first time. Additionally, we propose a RAG to accelerate the training process of the Meta-RL algorithm, along with the development of a selection module that ensures closed-loop control of the RAG. Extensive simulations and physical experiments have substantiated the superiority of the proposed method. In future research, we plan to explore fault-tolerant motion control strategies for more challenging terrains. Additionally, we will consider incorporating navigation strategies to enhance the overall performance of the robot.

References

[1] C. D. Bellicoso et al., "Advances in real-world applications for legged robots," *J. Field Robot.*, vol. 35, no. 8, pp. 1311–1326, 2018.

[2] J. Hooks et al., "ALPHRED: A multi-modal operations quadruped robot for package delivery applications," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5409–5416, Oct. 2020.

[3] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, Nov. 2006.

[4] S. Koos, A. Cully, and J.-B. Mouret, "Fast damage recovery in robotics with the T-resilience algorithm," *Int. J. Robot. Res.*, vol. 32, no. 14, pp. 1700–1723, Dec. 2013.

[5] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[6] C. F. Pana, I. C. Resceanu, and D. M. Patrascu, "Fault-tolerant gaits of quadruped robot on a constant-slope terrain," in *Proc. IEEE Int. Conf. Autom., Quality Test., Robot.*, Sep. 2008, pp. 222–226.

[7] J. Cui, Z. Li, J. Qiu, and T. Li, "Fault-tolerant motion planning and generation of quadruped robots synthesised by posture optimization and whole body control," *Complex Intell. Syst.*, vol. 8, no. 4, pp. 2991–3003, Aug. 2022.

[8] J.-M. Yang, "Fault-tolerant gaits of quadruped robots for locked joint failures," *IEEE Trans. Syst. Man Cybern., C, Appl. Rev.*, vol. 32, no. 4, pp. 507–516, Nov. 2002.

[9] J.-M. Yang, "Crab walking of quadruped robots with a locked joint failure," *Adv. Robot.*, vol. 17, no. 9, pp. 863–878, Jan. 2003.

[10] J.-M. Yang, "Kinematic constraints on fault-tolerant gaits for a locked joint failure," *J. Intell. Robotic Syst.*, vol. 45, no. 4, pp. 323–342, Apr. 2006.

[11] A. Nagabandi et al., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," 2018, *arXiv:1803.11347*.

[12] C. Chen et al., "Fast adaptation dynamics model for Robot's damage recovery," in *Proc. IEEE Int. Conf. Real-time Comput. Robot. (RCAR)*, Jul. 2022, pp. 45–50.
[13] Y. Seo, K. Lee, I. C. Gilaberte, T. Kurutach, J. Shin, and P. Abbeel, "Trajectory-wise multiple choice learning for dynamics generalization in reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 12968–12979.
[14] W. Okamoto, H. Kera, and K. Kawamoto, "Reinforcement learning with adaptive curriculum dynamics randomization for fault-tolerant robot control," 2021, *arXiv:2111.10005*.
[15] K. Lee, Y. Seo, S. Lee, H. Lee, and J. Shin, "Context-aware dynamics model for generalization in model-based reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5757–5766.
[16] D. Liu, T. Zhang, J. Yin, and S. See, "Saving the limping: Fault-tolerant quadruped locomotion via reinforcement learning," 2022, *arXiv:2210.00474*.
[17] X. Wu, W. Dong, H. Lai, Y. Yu, and Y. Wen, "Adaptive control strategy for quadruped robots in actuator degradation scenarios," in *Proc. 5th Int. Conf. Distrib. Artif. Intell.*, Nov. 2023, pp. 1–13.
[18] X. Chen, C. Qi, F. Gao, X. Tian, X. Zhao, and H. Yu, "Fault-tolerant gait a quadruped robot with partially fault legs," in *Proc. UKACC Int. Conf. Control (CONTROL)*, Jul. 2014, pp. 509–514.
[19] M. M. Gor, P. M. Pathak, A. K. Samantaray, J.-M. Yang, and S. W. Kwak, "Fault accommodation in compliant quadruped robot through a moving appendage mechanism," *Mechanism Mach. Theory*, vol. 121, pp. 228–244, Mar. 2018.
[20] C. Yu, W. Zhang, H. Lai, Z. Tian, L. Kneip, and J. Wang, "Multi-embodiment legged robot control as a sequence modeling problem," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 7250–7257.
[21] L. Chen et al., "Decision transformer: Reinforcement learning via sequence modeling," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 15084–15097.
[22] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 1126–1135.
[23] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018, *arXiv:1803.02999*.
[24] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–11.
[25] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast reinforcement learning via slow reinforcement learning," 2016, *arXiv:1611.02779*.
[26] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," 2017, *arXiv:1707.03141*.
[27] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 5331–5340.
[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
[29] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," 2017, *arXiv:1703.00810*.
[30] D. J. Hyun, S. Seok, J. Lee, and S. Kim, "High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the MIT Cheetah," *Int. J. Robot. Res.*, vol. 33, no. 11, pp. 1417–1445, Sep. 2014.
[31] I. M. A. Nahrendra, B. Yu, and H. Myung, "DreamWaQ: Learning robust quadrupedal locomotion with implicit terrain imagination via deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 5078–5084.
[32] Y. Bengio, J. Louradour, and R. Collobert, "Curriculum learning," in *Proc. Int. Conf. Mach. Learn.*, Aug. 2009, pp. 41–48.
[33] X. Cheng, Y. Ji, J. Chen, R. Yang, G. Yang, and X. Wang, "Expressive whole-body control for humanoid robots," 2024, *arXiv:2402.16796*.
[34] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Sci. Robot.*, vol. 5, no. 47, Oct. 2020, Art. no. eabc5986.
[35] X. Chen, C. Wang, Z. Zhou, and K. Ross, "Randomized ensembled double Q-learning: Learning fast without a model," 2021, *arXiv:2101.05982*.
[36] M. Bloesch et al., "State estimation for legged robots-consistent fusion of leg kinematics and IMU," *Robotics*, vol. 17, pp. 17–24, Jul. 2013.

Ci Chen received the B.E. degree in agricultural mechanization and automation from Northeast Agricultural University, Harbin, China, in 2018. She is currently pursuing the Ph.D. degree in control science and engineering with the Department of Control Science and Engineering, Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou, China.

Her research interests include deep reinforcement learning and legged robots.

Chao Li received the Ph.D. degree from the Department of Control Science and Engineering, Zhejiang University, Hangzhou, China, in 2016.

He is currently the Chief Technology Officer of DeepRobotics Company. His current research interests include legged robotics and robot locomotion.

Haojian Lu (Member, IEEE) received the B.E. degree in mechatronic engineering from Beijing Institute of Technology, Beijing, China, in 2015, and the Ph.D. degree in robotics from the City University of Hong Kong, Hong Kong, in 2019.

He is currently a Professor with the State Key Laboratory of Industrial Control and Technology and the Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou, China. His research interests include micro/nanorobotics, bioinspired robotics, medical robotics, micro aerial vehicles, and soft robotics.

Yue Wang (Member, IEEE) received the Ph.D. degree in control science and engineering from the Department of Control Science and Engineering, Zhejiang University, Hangzhou, China, in 2016.

He is currently a Professor with the Department of Control Science and Engineering, Zhejiang University. His current research interests include mobile robotics and robot perception.

Rong Xiong (Senior Member, IEEE) received the Ph.D. degree in control science and engineering from the Department of Control Science and Engineering, Zhejiang University, Hangzhou, China, in 2009.

She is currently a Professor with the Department of Control Science and Engineering, Zhejiang University. Her current research interests include motion planning and SLAM.