IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 September 2023, accepted 4 October 2023, date of publication 12 October 2023, date of current version 16 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3324039

RESEARCH ARTICLE

A Hierarchical Robot Learning Framework for Manipulator Reactive Motion Generation via Multi-Agent Reinforcement Learning and Riemannian Motion Policies

YULIU WANG[1,2], RYUSUKE SAGAWA[1,2], (Member, IEEE), AND YUSUKE YOSHIYASU[2], (Member, IEEE)

[1]Intelligent and Mechanical Interaction Systems Program, University of Tsukuba, Tsukuba, Ibaraki 305-8577, Japan
[2]Computer Vision Research Team, Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology, Tsukuba, Ibaraki 305-8560, Japan

Corresponding author: Yuliu Wang (wangyuliu.o@aist.go.jp)

This work was supported by JST SPRING (Japan Since and Technology Agency Support for Pioneering Research Initiated by the Next Generation), Grant Number JPMJSP2124, JSPS JP22H00545, JP22H05002 (Japan Society for the Promotion of Science) and NEDO JPNP20006 (New Energy and Industrial Technology Development Organization) in Japan.

ABSTRACT Manipulators motion planning faces new challenges as robots are increasingly used in dense, cluttered and dynamic environments. The recently proposed technique called Riemannian motion policies(RMPs) provides an elegant solution with clear mathematical interpretations to such challenging scenarios. It is based on differential geometry policies that generate reactive motions in dynamic environments with real-time performance. However, designing and combining RMPs is still a difficult task involving extensive parameter tuning, and typically seven or more RMPs need to be combined by using RMPflow to realize motions of a robot manipulator with more than 6 degrees-of-freedoms, where the RMPs parameters have to be empirically set each time. In this paper, we take a policy to decompose such complex policies into multiple learning modules based on reinforcement learning. Specifically, we propose a three-layer robot learning framework that consists of the basic-level, middle-level and top-level layers. At the basic layer, only two base RMPs i.e. target and collision avoidance are used to output reactive actions. At the middle-level layer, a hierarchical reinforcement learning approach is used to train an agent that automatically selects those RMPs and their parameters based on environmental changes and will be deployed at each joint. At the top-level layer, a multi-agent reinforcement learning approach trains all the joints with high-level collaborative policies to accomplish actions such as tracking a target and avoiding obstacles. With simulation experiments, we compare the proposed method with the baseline method and find that our method effectively produces superior actions and is better at avoiding obstacles, handling self-collisions, and avoiding singularities in dynamic environments. In addition, the proposed framework possesses higher training efficiency while leveraging the generalization ability of reinforcement learning to dynamic environments and improving safety and interpretability.

INDEX TERMS Riemannian motion policies, motion generation, motion planning, robot learning, multi-agent reinforcement learning, hierarchical reinforcement learning.

I. INTRODUCTION

Motion planning is a fundamental and critical technology for robots, and it is typically faced with the challenge of balancing efficiency with success rate and safety. This challenge becomes even more pronounced in non-structured, dynamically changing environments. In the past, sample-based methods have been used to generate feasible trajectories and ensure safety [2], [13], and these methods have been widely applied in stable environments such as factories.

The associate editor coordinating the review of this manuscript and approving it for publication was Laura Celentano.

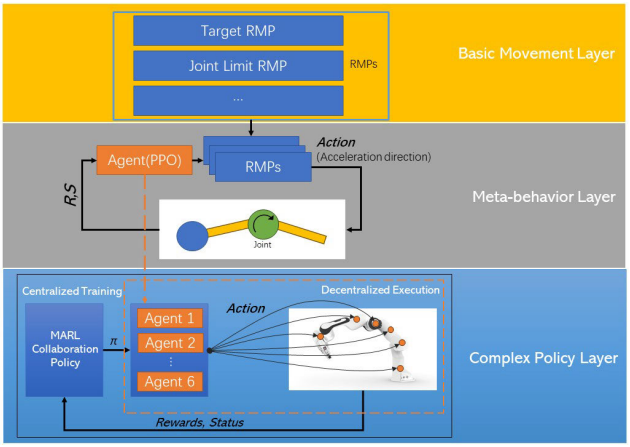


FIGURE 1. We utilize two RMPs as basic-layer actions for output stability(yellow). At the meta-behavioral layer, we pre-train for agents, by employ hierarchical structure for real-time RMP selection to ensure the interpretability of actions (gray). Our system consists of six agents, each corresponding to a joint of the manipulator. Agent operates under the framework of multi-agent reinforcement learning with centralized training and decentralized execution(blue), where the local agent observes the local state and the central cooperation network considers the global state and policy. During testing, the agents generates action outputs based on the observed state.

However, in dynamic environments where information is not completely known, sampling may fail to cover sudden situations. In order to adapt to the environment, some online planning methods perform real-time sampling and re-planning, as the dimension of robot actions increases, the computational cost also increases significantly, which may lead to a delay in control and decreased safety [7]. Furthermore, traditional motion planning methods often rely on an accurate model of the environment and dynamics, which may not be feasible in real-world scenarios. These issues have motivated the development of new approaches to motion planning that are more adaptable to dynamic and uncertain environments.

Reactive motion generation is a collision avoidance algorithm that can quickly respond locally. Among them, the Riemannian motion policies [25] is impressive, which considers the task space as a Riemannian manifold, configures the robot as a point on the manifold, and models it as a second-order dynamical system through differential equations, where the differential equation is the motion policy. The policy maps position and velocity to acceleration and is coupled with the corresponding Riemannian metric, which can capture the important directions for the policy in space. This method transforms nonlinear tasks from one smooth manifold to another in a geometrically consistent manner, thereby having linear characteristics and allowing the rapid solution of joint accelerations induced by dynamic environments that are nonlinear. The Riemannian motion policy has been successfully applied to many robot tasks. Riemannian Motion Policies Flow (RMPflow) [34] is a policy-fusion framework based on differential geometry, specifically designed to synthesize motion policies within the

TABLE 1. The table shows the difference in the number of policies used by our method and RMPflow for a 6 degrees of freedom (DoF) manipulator. For our method, In addition to the two basic RMP, other policies will be equivalent to the complex collaborative policies generated by our training framework.

	RMPflow	Ours
Target RMP	✓	✓
Collision RMP	✓	✓
Axis target RMP	✓	MARL-based Training Framework
Joint limit RMP	✓	
Damping RMP	✓	
C-space target RMP	✓	
Joint Velocity Limit RMP	✓	

Riemannian Motion Policies framework. Its primary goal is to address the challenge of combining motion policies from different paradigms consistently and optimally in robotic systems. RMPflow leverages differential geometry to ensure geometrically consistent transformations of RMPs, enabling the fusion of diverse motion techniques into a unified and coherent motion policy within the configuration space. The strengths of RMPflow lie in its ability to provide a robust framework for integrating motion policies from different paradigms in a consistent and optimal manner. By using differential geometry, RMPflow guarantees that the transformations of RMPs maintain geometric consistency, enhancing the overall reliability of the combined motion policies.

However, it should be noted that while the Riemannian motion policy has shown successful applications in various Manipulator tasks [32], its design itself is a complex task. For manipulators, it requires the incorporation of multiple Riemannian motion policies, including goals, C-space goals, obstacle avoidance, joint constraints, and more. Additionally, the need for RMPflow to merge multiple sub-policies involves significant parameter tuning, which can be time-consuming and resource-intensive. Careful attention and effort are necessary to effectively design and tune these Riemannian motion policies to achieve optimal performance in different robotic applications. Hence, the rapid design and deployment of RMPflow for a Manipulators remain challenging tasks.

Deep learning has indeed shown successful applications in simplifying parameter design in various domains [36]. As a result, researchers have been exploring the combination of Riemannian Motion Policies and deep learning (DL) with the aim of simplifying the design of RMPs and enhancing their robustness [35]. However, one of the major challenges faced in this pursuit, similar to other DL applications, is the collection of training data [21]. Training deep learning models typically requires large amounts of data, which may not always be readily available in the context of RMPs. Some approaches collect data by mimicking human behavior and training the model to learn the RMP design [41], [42], but it is also necessary to learn RMPs separately for different tasks and combine them with RMPflow, and collecting human behavior in large quantities can be costly. The scarcity of

training data can hinder the integration of DL and RMP, posing difficulties in achieving the desired simplification and robustness. Some studies have attempted to combine Deep Reinforcement Learning (DRL) with RMPflow to tackle more complex tasks [37], [38]. In these efforts, RMPs are called modularly within the DRL framework to accomplish specific tasks. While this approach allows for the utilization of RMPs in complex scenarios, it does not fundamentally simplify the design of RMPs, in this context may be limited to specific task optimization rather than fundamentally improving the design process.

A. RESEARCH OBJECTIVES AND CONTRIBUTIONS

The principal aim of this study is to utilize the design process of a comprehensive framework based on RMPs and DRL to supplant the intricate RMPflow design procedure with this new approach. Through the integration of deep reinforcement learning, we achieve a heightened level of trainability, adaptability, and robustness in the RMP motion generation framework, thereby enabling it to effectively adapt to intricate and dynamic environments.

To this end, we propose a three-layer robot learning framework, shown in FIGURE 1. In this study, we streamline the set of seven prevalent RMPs to a more concise pair within the lower layer. Subsequently, by employing hierarchical reinforcement learning, we achieve real-time pre-training of a joint by implementing the strategies associated with these two RMPs. Ultimately, the pre-trained agents are effectively deployed at each joint, facilitating the training of collaborative strategies through multi-agent reinforcement learning, thereby enabling the successful execution of intricate tasks. The correspondence between our framework and RMPflow can be found in TABLE 1.

With this structural design, we assign complexity to the layers in order to avoid optimization in the high-dimensional action space. While retaining the advantages of reinforcement learning, we are able to simplify the system design and improve the training speed, while ensuring efficiency and safety of motion generation.

II. RELATED WORK

A. RIEMANNIAN MOTION POLICY AND RMPFLOW

Differential geometry provides a powerful framework for effectively describing and analyzing the fundamental characteristics of manipulators. Motion generation and control challenges can be conceptualized as the process of mapping desired behaviors from the task space T, consisting of one or multiple smooth manifolds, to the configuration space C, another smooth manifold. This mapping is represented by a differential map $\phi : C \to T$, known as the task map. By equipping these manifolds with a Riemannian metric M, by establish meaningful connections between geometric properties, such as angles and distances. This allows for the design of a curve $q(t) \in C$ that captures the desired behavior of the task space T in a concise and elegant manner. The Riemannian metric facilitates the measurement and comparison of geometric properties, enabling the definition of a well-defined trajectory in the configuration space corresponding to the desired behavior in the task space.

In the context of motion policies, a Riemannian Motion Policy is a representation of motion defined on a Riemannian manifold. An RMP [25] is typically expressed as $(a, M)_m$ or $(f, M)_m$, depending on the specific formulation. In the canonical form $(a, M)_m$, the RMP is represented by a second-order dynamical system, denoted as $a : x \mapsto \ddot{x}^d \in \mathbb{R}^m$, that maps the generalized coordinates x and their derivatives $\dot{x}$ to desired accelerations $\ddot{x}$. The Riemannian metric $M = M(x, \ddot{x}) \in \mathbb{R}^{m \times m}$, which varies smoothly with the state $(x, \dot{x})$, governs the weighting of the RMP and can be interpreted as an inertial matrix. Alternatively, in the natural form $(f, M)_m$, the RMP is defined as a mapping from position and velocity to the desired force, represented by f. The force mapping f and the acceleration mapping a in the canonical form are related through $f = Ma$. The natural form is commonly used for space transformations.

Within the RMP framework, operators such as push, pull, and add are provided. The push and pull operators allow the transformation of an RMP defined in one task space to another, leveraging the task map ϕ and its Jacobian J. On the other hand, the add operator is used to compose RMPs defined within the same task space into a unified policy. These operators enhance the versatility and adaptability of the RMP framework, enabling the manipulation of RMPs to achieve desired behaviors and tasks.

RMPflow is a computational graph tailored for the automatic generation of motion strategies. At the heart of RMP is the encapsulation of non-Euclidean behavior as a dynamical system in an inherently nonlinear task space. As a geometric framework, RMPflow is proficient at generating multi-task motion strategies, providing a powerful tool for seamlessly generating motion strategies for robotic entities and other autonomous systems. The strength of RMPflow lies in its inherent adaptability and flexibility, which does not require the user to design a globally optimal policy, but rather allows for combining sub-task strategies to solve complex tasks, such as in cluttered environments. Motion can be decomposed into proximity and obstacle avoidance.RMPflow enables the user to design strategies for a wide range of tasks and environments, but it is worth noting that it needs to be based on reasonably well-designed localized strategies.

B. HIERARCHICAL REINFORCEMENT LEARNING FOR ROBOTIC MOTION POLICIES FUSION

Recent studies suggest that deep reinforcement learning (DRL) [16] has potential for complex problem-solving [26], like in computer games, Go, and Bridge. Researchers are exploring RL for manipulator motion planning [9], [18]. By treating the motion planning as a Markov decision process (MDP) and maximizing expected rewards, RL helps the manipulator make decisions based on its state, actions, and

environment's stochastic nature [17], [22], [29]. Despite its benefits, RL faces challenges in manipulator motion planning due to high-dimensional state spaces with six or more degrees of freedom and low sampling efficiency [24].

Finding the optimal policy for manipulator movement using reinforcement learning proved to be a formidable task [3]. In earlier attempts, discrete states and action values were maintained within a Q table, but in the case of redundant manipulator systems, typically comprising 6 or more degrees of freedom (DOF), the actions have continuous values, thereby rendering the system endowed with nearly infinite states. Hierarchical Reinforcement Learning (HRL) [23] has emerged as a promising approach to address these challenges [10]. HRL aims to decompose a complex task into a hierarchy of sub-tasks, each of which can be learned independently and then combined to solve the overall task. The main difference between RL and HRL lies in their underlying learning frameworks. RL learns a single policy that maps states to actions, whereas HRL learns a hierarchy of policies that map states to sub-tasks by decomposing complex tasks into simpler sub-tasks [30], [33]. RL struggles with scaling due to the exponential growth of the state and action spaces, but HRL overcomes this limitation by operating at multiple levels of abstraction, enabling more efficient learning. Furthermore, HRL enables transfer learning by reusing sub-policies, or low-level policies, for different tasks [28]. This transfer of knowledge between sub-tasks can enhance learning efficiency and generalization capabilities. Another advantage of HRL is improved exploration. By utilizing high-level actions, HRL reduces the action space, enabling more effective exploration of the environment [31]. High-level actions provide a higher-level control mechanism, guiding the agent to explore promising regions of the state space. HRL has the potential to improve the efficiency and effectiveness of RL, making it a valuable research direction in the field of robotics.

Robotics extensively employs the practice of abstracting motion generation problems into multiple decision-making layers, a well-established approach, particularly for intricate robotic tasks involving extended time horizons. This hierarchical decision-making paradigm encompasses two fundamental facets: the utilization of multi-level planners and the operation within the parameter space of motion policies. Multi-level planners, exemplified by hierarchical planning hierarchical RL, generate sub-goals that an underlying planner or policy must strive to achieve. On the other hand, methods falling under the latter category focus on the fine-tuning of constraint functions of dynamic motion primitives or selecting a policy from a diverse ensemble of experts. However, this selective behavior becomes problematic in scenarios involving unexpected environmental changes, leading to either sub-optimal performance or a fusion of experts with complexly encoded behaviors. To counteract these challenges, we propose that the composition of simple and stable reactive policies can give rise to intricate reactive behaviors in robots. This process uses a mixture of high-level experts, who then proceed to select from a lower-level repertoire of primitive policies tailored to cater to long-term robotic tasks.

C. MULTI-AGENT REINFORCEMENT LEARNING

Multi-agent reinforcement learning (MARL) represents a prominent approach for decomposing intricate action spaces. Diverging from the hierarchical reinforcement learning (HRL) paradigm, MARL does not explicitly entail an evident hierarchical relationship among agents. Instead, MARL centers its attention on investigating the behavior of multiple learning agents operating within a mutually shared environment [12]. MARL incorporates social metrics to evaluate cooperation, reciprocity, equity, social influence, and compete among coexisting agents, resulting in complex group dynamics [19]. MARL draws upon game theory and multi-agent systems to understand agent interactions and policies. By combining reward optimization with sociological concepts, MARL offers a unique perspective on agent behaviors and collective performance. MARL's advantages include facilitating collaboration among agents, enhancing robustness and adaptability to changes in the environment and other agents. It also scales to complex problems [27] by leveraging collective intelligence and computational resources.

In robotics, When multiple agents interact with an environment concurrently, the entire system assumes the form of a multi-agent system. Each agent adheres to the objective of reinforcement learning, namely, maximizing the cumulative returns attainable. During this process, the global state of the environment undergoes changes that are contingent upon the joint actions executed by all agents. Consequently, when learning agent policies, it becomes imperative to account for the influence exerted by these joint actions.

Interestingly, some studies have provided some different perspectives on the fact that a multi-agent system does not necessarily intuitively exist independently of each other in an environment. An individual with a continuous structure that also has multiple moving bodies with independent degrees of freedom of movement can also be considered an agent [4]. An individual with multiple such bodies that move in concert can also be considered a multi-agent system. Such systems are also found in nature, a clear example being the cephalopod octopus [39], which has a distributed nervous system, a central nervous system, and a distributed nervous system on eight carpal legs, each of which has a certain degree of autonomy, while the central nervous system is responsible for handling the coordinated movements between the carpal legs. Each joint of the manipulator can be considered an independent agent and a complete manipulator can be redefined as a multi-agent system, and multi-agent reinforcement learning can be used to solve the challenge of motor problems.

In contrast to single-agent reinforcement learning and Markov Decision Processes, multi-agent reinforcement

learning operates within the framework of a Markov game. Within this context, the Markov game involves multiple agents, each independently selecting and executing actions simultaneously, guided by the current state of the environment. The joint actions of all agents collectively influence the state transition and subsequent updates of the environment. Formally, the MDP game is defined as a tuple $(S, A_1 \ldots A_i, T, R_1 \ldots R_i, \gamma)$, wherein S denotes the set of states, A_i and R_i respectively represent the action set and reward set pertaining to agent i, and T signifies the transition probability of the environmental state, along with the discount factor. Accordingly, the expectation of agent i concerning the cumulative reward obtained through the utilization of policy π^i can be mathematically expressed as follows:

$$V^{i}_{\pi^i,\pi^{-i}} := E\left[\sum_{t\geq 0}\gamma^t R^i(s_t, a_t, s_{t+1})|a^i_t \sim \pi^i(\cdot|s_t), s_0 = s\right] \tag{1}$$

In the realm of multi-agent reinforcement learning, the central aim is to reach a state of Nash equilibrium across all participating agents. This entails that within the cooperative domain of multiple agents, no individual agent can attain superior cumulative returns by deviating from its current policy. It is crucial to emphasize that the Nash equilibrium does not inherently align with the global optimum, yet it stands as the most probable outcome and signifies a state that is prone to convergence during the process of learning.

Multi-agent reinforcement learning encompasses three types of tasks: adversarial, cooperative, and mixed. In the majority of multi-robot tasks, cooperation constitutes the primary task category. From a different perspective, modeling multi-agent systems based on cooperation for a single manipulator can simplify the state dimension of the redundant manipulator. ''Cooperation'' implies that multiple agents must collaboratively accomplish a shared task, thereby rendering the achievement of the goal contingent upon the collective behavior of the individual agents. If an agent pursues an independent course of action, it becomes challenging for them to effectively cooperate with other teammates and collectively achieve high returns. Consequently, the policy learning of agents must still consider the impact of joint actions and the influence exerted by other decision-making agents.

The essence of the matter lies in the collaboration mechanism, which can manifest in either a decentralized or centralized manner. Under a decentralized collaboration mechanism, each agent retains its autonomy in selecting actions. Although this approach reduces the dimensionality of the action space, it results in the simultaneous execution of multiple individual agents. Consequently, as the number of agents within the environment increases, the system's inherent randomness also escalates, giving rise to safety concerns in manipulator tasks. Conversely, the centralized collaboration mechanism dictates a uniform action policy for all agents. This enhances system stability but curtails the autonomy of each agent, leading to sub-optimal results.

III. METHODOLOGY

A. OVERVIEW

In this section, we introduce our three-layer training framework. The bottom layer employs RMPs to provide reactive motor policies for local actions, and we will elaborate on the specifics of these RMPs. In the middle layer, we employ hierarchical reinforcement learning combined with RMPs to enable the simultaneous execution of multiple RMPs without relying on RMPflow. Finally, at the high level, joint policies are deployed and multi-agent reinforcement learning is utilized to coordinate the movements of all joints, enabling the accomplishment of more complex behaviors. We will also provide the training details of this comprehensive framework.

Starting with the bottom layer, RMPs are utilized to generate reactive motion policies for local actions. These policies are designed to enable quick and adaptive responses to the environment. The details of these RMPs, including their formulation and implementation, will be presented, showcasing their effectiveness in providing fine-grained control and precise motor actions. At the middle layer, hierarchical reinforcement learning is combined with RMPs to facilitate the simultaneous execution of multiple RMPs. This hierarchical approach allows for the coordination and integration of different RMPs without relying on RMPflow, resulting in more flexible and scalable control. The training process and architecture of this hierarchical reinforcement learning framework will be described in detail, highlighting how it enables the efficient utilization of RMPs in complex scenarios. At the high level, policies are deployed for each joint, and multi-agent reinforcement learning is employed to control all joints, ensuring coordinated movements and achieving more sophisticated behaviors. The training methodology and techniques utilized within this high-level framework will be presented, demonstrating how multi-agent reinforcement learning enables joint coordination and enhances the overall performance of the robotic system.

Our three-layer training framework combines reactive motion policies through RMPs, hierarchical reinforcement learning, and multi-agent coordination. Each layer contributes to the overall control and behavior of the robotic system, allowing for effective and adaptive motion planning and execution.

B. BASIC MOVEMENT LAYER: RMP DESIGN

In terms of functions, the base movement layer addresses the dynamics of the local joints of the manipulator, it determines how much acceleration or force is applied when the robot moves. In the robot configuration space, we define the joint angles as q, and hence the joint velocities and accelerations as: $\dot{q}$, $\ddot{q}$. Similarly, in the nonlinear task space, we define the joint positions as x and the velocity and accelerations as $\dot{x}$, $\ddot{x}$. We formulate a motion policy $\pi : \mathbf{q}, \dot{\mathbf{q}} \mapsto \ddot{\mathbf{q}}$ as

a dynamical system (second-order differential equation) that maps position and velocity to acceleration. For the nonlinear task space, we can analogously write $\pi : \mathbf{x}, \dot{\mathbf{x}} \mapsto \ddot{\mathbf{x}}$ without loss of generality. The acceleration can be expressed in terms of direction, and readily in terms of forces or moments as a general dynamics formulation. As we will demonstrate, representing the policy in terms of forces or moments is often more expedient, since it is more natural to define all motion policies in the task space. The configuration space and task space are linked by differentiable mappings ϕ. If we denote the task map ϕ as Jacobian.

$$\mathbf{J}_\phi \equiv \frac{\partial \phi}{\partial \mathbf{q}} \in \mathbb{R}^{k \times d} \tag{2}$$

then the task space velocities and accelerations can be expressed as follows:

$$\begin{aligned} \dot{\mathbf{x}} &= \frac{d}{dt}\phi(\mathbf{q}) = \mathbf{J}_\phi \dot{\mathbf{q}} \\ \ddot{\mathbf{x}} &= \frac{d^2}{dt^2}\phi(\mathbf{q}) = \mathbf{J}_\phi \ddot{\mathbf{q}} + \dot{\mathbf{J}}_\phi \dot{\mathbf{q}} \approx \mathbf{J}_\phi \ddot{\mathbf{q}} \end{aligned} \tag{3}$$

The dynamic equation has the following form, which is derived from a refinement of the Euler-Lagrange equation, and introduce a velocity term $\dot{x}$.

$$\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} = -\mathbf{K}\left(\mathbf{x} - \mathbf{x}_g\right) - \mathbf{B}\dot{\mathbf{x}} \tag{4}$$

Equation be solved in Riemannian manifold, which is defined by Riemannian metric $\mathbf{M}$, which defines the curvature of the space ($\mathbf{M}$ can be artificially designed to fit the specific task), Therefore it is also called inertia matrix. $\mathbf{C}$ is the Coriolis force term, which together with the position $\mathbf{x}$ creates the Coriolis force (the force created by the curvature of space). $\mathbf{K}$ is the stiffness matrix, $\mathbf{B}$ is the damping matrix, and $\mathbf{x_g}$ is the goal in manifold. Moving through the curvature of the defined space determines the direction in which acceleration is most important to the task for efficient motion.

We employ two simple RMPs in this study: the Target RMP and the Collision Avoidance RMP. The fundamental nature of these two policies allows for the foundational motion control of individual joints. The reason we chose these two RMPs is these are the most fundamental maneuvers and the most potentially hazardous in a robotic navigation task. As a basic skill, one should focus on the most important tasks for quick, reactive movement, leaving other tasks to higher levels of planning is a better option. We also hope to minimize the number of RMPs to make the ensuing hierarchical reinforcement learning easier to train. By synergistically combining the motions of multiple joints, intricate and sophisticated motions can be accomplished. Moreover, the inherent adjustability of these two RMPs and their independence from robot-specific parameters bestow them with a considerable degree of flexibility, enhancing their applicability across diverse robotic systems. These policies provide the robot with fundamental motion outputs, enabling it to approach the target position and avoid collision.

Regarding the Target RMP, the following expression governs its behavior:

$$\ddot{x} = k_p(x_0 - x)/(\|x_0 - x\| + \epsilon) - k_d|\dot{x}|\dot{x} \tag{5}$$

here, k_p represents the position gain, $-k_d|\dot{x}|\dot{x}$ is the damping RMP, k_d signifies the damping gain. Especially, we found that in the context of tracking a dynamic target, the adoption of undamped Riemannian Motion Policies will lead to recurring oscillations in the system's behavior. Nevertheless, the incorporation of damped RMPs significantly mitigates this issue. ϵ controls the length scale that governs the transition between the constant acceleration region distant from the target and the linear region near the target. Each motion policy necessitates the application of a distinct Riemannian metric denoted as M. To acquire a comprehensive understanding of the derivation of the Riemannian metric, we recommend consulting the original paper [25]. For the target RMP, the Riemannian metric is designed as follows:

$$M = [\beta(x)b + (1 - \beta(x))]\,[\alpha(x)M_{\text{near}} + (1 - \alpha(x))M_{\text{far}}] \tag{6}$$

The metric denoted as M, also referred to as the inertia matrix, resides within the spectrum delineated by the rank-deficient metric S and the identity matrix I. Specifically, the metric S exerts a pronounced influence in regions distanced from the target, The identity matrix I assumes dominance in close proximity to the target, thereby facilitating expedited convergence. The blending of these matrices is governed by a radial basis function, particularly a Gaussian function. This function gradually transitions from a minimum constant value in regions distant from the target to a value approximating 1 near the target.

In the aforementioned equations, where,

$$\begin{aligned} \alpha(x) &= (1 - \alpha_{\min}) \exp\left(\frac{-||x_0 - x||^2}{2\sigma_a^2}\right) + \alpha_{\min} \\ \beta(x) &= \exp\left(-\frac{||x_0 - x||^2}{2\sigma_b^2}\right) \\ M_{\text{near}} &= \mu_{\text{near}} I \\ M_{\text{far}} &= \mu_{\text{far}} S = \frac{\mu_{\text{far}}}{||x_0 - x||^2}(x_0 - x)(x_0 - x)^T \end{aligned} \tag{7}$$

σ_a is the Length scale of the Gaussian controlling blending between S and I, σ_b is the Length scale of the Gaussian controlling boosting near the target, and the unit is m. The parameter $\alpha_{\min}$ governs the minimal impact of M_{near} on the overall metric M. Additionally, μ_{near} and μ_{far} serve as priority weights dictating the significance of the contributions from M_{near} and M_{far} to the composite metric M. Meanwhile, the scaling factor denoted as b exerts influence by regulating the strengthening intensity in close proximity to the target.

For RMP aimed at collision avoidance, the objective is to prevent collisions between the joint and the surrounding obstacles or other joints. To enhance computational efficiency, we approximate the geometric properties of each

link using capsules and simplify the problem of link-to-link collision avoidance by focusing on evading collisions between the closest points on the capsules, referred to as witness points. Consequently, only a single collision RMP is required for each pair of links. We adopt the algorithm proposed in RMP to determine the position of the witness points. The collision avoidance RMPs employed in this study are defined within the one-dimensional Euclidean distance space, and they are specified by the repulsive acceleration policy and metric, respectively.

$$\begin{aligned} \ddot{x}(x,\dot{x}) &= k_p \exp\left(-x/l_p\right) - k_d \frac{\sigma(\dot{x})\dot{x}}{x/l_d + \epsilon_d} \\ m(x,\dot{x}) &= \sigma(\dot{x}) g(x) \frac{\mu}{x/l_m + \epsilon_m} \end{aligned} \tag{8}$$

where,

$$\begin{aligned} \sigma(\dot{x}) &= 1 - \frac{1}{1+\exp\left(-\dot{x}/v_d\right)} \\ g(x) &= \begin{cases} x^2/r^2 - 2x/r + 1, & x \le r \\ 0, & x > r \end{cases} \end{aligned} \tag{9}$$

In the aforementioned equations, the variable x denotes the Euclidean distance between the witness points of the capsules, and $\dot{x}$ represents the rate of change of x. The parameters $k_p\left[\mathrm{m/s^2}\right]$ and $k_d\left[s^{-1}\right]$ correspond to the repulsion and damping gains, respectively. The parameter μ determines the relative priority of the collision RMP in relation to other RMPs, Here we make $\mu = 1$, because in the next layer, the weights are distributed end-to-end by RL. $r[m]$ controls the distance at which the collision RMP is deactivated. The scaling parameters are denoted as $l_p[m]$, $l_m[m]$, and $v_d[m/s]$. The parameters ϵ_m and ϵ_d serve as offset parameters.

C. META-BEHAVIOR LAYER: HIERARCHICAL REINFORCEMENT LEARNING METHOD FOR RMP SELECTION

At this layer we aim to replace the pullback operation [34] in RMPflow with an end-to-end approach to integrate the two RMPs, The motion exhibited by a manipulator can be conceptualized as a system that adheres to the Markov property, wherein the subsequent state S_{t+1} solely relies on the current state S_t, independent of any preceding states. For the manipulator, MDP can be represented as a tuple (S, A, P, R, γ). Here, S encompasses the state space of the manipulator, encompassing essential parameters like joint rotation angles, Cartesian coordinates of the end effector, and base position, among others. A denotes the set of possible manipulator motions, encapsulating the rotational range of each joint motor. For the purpose of this study, we consider the discrete RMP option, the sequential execution of two RMPs denoted a and b. The state transition function P is established based on the manipulator's forward kinematics. Additionally, we define the reward function R to quantify the desirability of different states, and γ is the discount factor that plays a vital role in balancing the learning system's inclination towards immediate rewards with its exploration of long-term rewards.

By utilizing reinforcement learning techniques, such an MDP can determine an optimal policy, denoted as $\pi(a \mid s) = p(A_t = A \mid S_t = S)$, which enables the selection of actions based on observations to maximize the reward R. The action-value function $Q_\pi(s, a)$ is employed to determine the maximum reward, which can be expressed as follows:

$$Q_\pi(s,a) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid A_t = [a,b], S_t = s\right] \tag{10}$$

In our system, each agent is modeled using a three-layer neural network with 128 neurons per layer. The Rectified Linear Unit (ReLU) activation function is applied to the first two layers. The network takes the cartesian coordinates of the joint as input and considers Riemannian Motion Policies (RMPs) at moment t as fundamental actions output. Similar to past hierarchical approaches [40], we utilize the notion of ''options'' to generalize primitive behaviors to temporally extended courses of action. An option is characterized by three key components: a policy π that maps states $\mathcal{S}$ and actions $\mathcal{A}$ to probabilities [0, 1], In this work, we consider Riemannian Motion Policies (RMPs) as fundamental actions and utilize the notion of ''options'' to generalize primitive behaviors to temporally extended courses of action. An option is characterized by three key components: a policy π that maps states $\mathcal{S}$ and actions $\mathcal{A}$ to probabilities [0, 1], a termination condition β that determines if the option should finish upon transitioning to a new state $s^+ \in \mathcal{S}^+$, and an initiation set $\mathcal{I} \subseteq \mathcal{S}$ defining states where the option can be applied. Concretely, an option $\langle \mathcal{I}, \pi, \beta \rangle$ is applicable in state s_t if and only if $s_t \in \mathcal{I}$. Once initiated, actions are selected according to π until the option terminates stochastically based on β. For a Markov option, the next action a_t is chosen by sampling the distribution $\pi(s_t, \cdot)$. The environment transitions to a new state s_{t+1}, where the option either terminates with probability $\beta(s_{t+1})$ or continues, prompting the selection of $a_{t+1} \sim \pi(s_{t+1}, \cdot)$. This process persists until termination conditions are satisfied at some future state $s_{t+2}, s_{t+3}, \ldots$. Upon completion of an option, the agent can initiate a new option, a termination condition β that determines if the option should finish upon transitioning to a new state $s^+ \in \mathcal{S}^+$, and an initiation set $\mathcal{I} \subseteq \mathcal{S}$ defining states where the option can be applied.

Through the consideration of a set of options, the initiation sets inherently establish a collection of available options $\mathcal{O}_s$ for each state $s \in \mathcal{S}$. Notably, these sets $\mathcal{O}_s$ bear a striking resemblance to the sets of available actions, conventionally denoted as $\mathcal{A}_s$. To establish a unifying framework for these two types of sets, it is noteworthy that actions can be regarded as a specific case of options. Each action a corresponds to an option that is available whenever a is available ($\mathcal{I} = s : a \in \mathcal{A}_s$), endures for precisely one step ($\beta(s) = 1, \forall s \in \mathcal{S}$), and selects a universally ($\pi(s,a) = 1, \forall s \in \mathcal{I}$). The agent's decision-making involves options encompassing

single-step and multi-step. Options bear a close resemblance to actions while accounting for temporal extension via well-defined termination conditions. This enables exploring sequences of options similar to sequences of actions.

Allows further analysis. a is initiated first and runs until termination, then b executes until it terminates or is skipped if a finishes outside b's initiation set. Combining these forms a new composite ab. Composing two Markov options typically results in a semi-Markov option since the policy changes before and after a terminate. Composing semi-Markov options forms another semi-Markov option, enabling complex multi-step behaviors through hierarchical composition.

D. COMPLEX POLICY LAYER: MULTI-AGENT REINFORCEMENT LEARNING FOR COLLABORATIVE POLICY TRAINING

At this layer, we deploy pre-trained agents to each joint and use MARL to train these agents. The complex policy layer can be viewed as the process of solving the inverse kinematics by modeling the problem as an MDP tuple $(X, \Delta Q, p, R)$ and fitting the numerical solution by using MARL, which X is The state represents the robot end-effector position and orientation in the task space. ΔQ is the action space is the set of small joint displacements in the joint space $\Delta \mathrm{Q} = \left[\Delta \mathrm{q1}, \ldots, \Delta \mathrm{qn}\right]$ where n is number of joints. p is the state transition function represents the forward kinematics mapping from joint space to task space i.e., $x_{t+1} = p\left(x_t, \Delta \mathrm{qt}\right)$. The reward function R evaluates the distance of end-effector from the desired position. $\pi\ (x)$ is the policy determines which joint displacement action to take in a given state x. The objective is to learn the policy $q\,(x)$ that minimizes the cumulative discounted reward and brings the joint to the target position. The policy essentially learns the inverse kinematics mapping from task space to joint space. Here, we use multi-agent reinforcement learning to optimize the policy.

Within dynamic environments, housing multiple agents, the policy gradient approach can make the convergence of the reward function difficult in the face of drastic changes in the environment. Assuming that the action space of the manipulator is discrete or 2-dimensional to alleviate this problem [11], [15]. Nevertheless, this assumption deviates considerably from the real-world environments in which the majority of robots operate. One of our previous studies showed that structured multi-agent systems using centralized training with decentralized execution(CTDE) are stably trained in the aforementioned case [43]. Multi-Agent Proximal Policy Optimization(MAPPO) [14] utilizes the actor-critic (AC) [6] architecture, the training process of the MAPPO algorithm entails an iterative approach to optimize the policies of multiple agents in a CTDE manner. It is aimed at enhancing multi-agent systems' performance by addressing challenges such as non-stationarity and credit assignment in such settings. The MAPPO algorithm leverages the Proximal Policy Optimization (PPO) [5] as its foundation, incorporating value function estimation and entropy regularization to foster both exploration and exploitation. During training, each agent observes the global state and interacts with its environment, generating trajectories then utilized to update their respective policies. Critically, MAPPO employs a trust region to ensure stable policy updates, thus preventing drastic policy deviations that could hinder convergence.

In the context of a scenario involving n agents, the network parameters of these agents are denoted as $\theta = \theta_1, \ldots, \theta_n$, and $\pi = \pi_1, \ldots, \pi_n$ represents the policy of each agent. The policy gradient update for each agent can be formulated as follows:

$$\nabla\theta_i J\left(\pi_i\right) = E_{x,a\sim D}\left[\nabla\theta_i \pi_i\left(a_i \mid o_i\right) \nabla_{a_i} Q_i^{\pi}\left(x, a_1, \ldots, a_N\right)\big|_{a_{i=\pi_i(o_i}}\right] \tag{11}$$

here, D represents the replay buffer containing data of $(x, x', a_1, \ldots, a_N, r_1, \ldots, r_N)$, where o_i denotes the observation of the i-th agent, and $x = [o_1, o_2, \ldots, o_i]$ represents the globally observed state information.

The action-value function is updated using the following equation:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{x,a,r,x'}\left[Q_i^{\pi}\left(x, a_1, \ldots, a_N\right) - y^2\right] \tag{12}$$

where y is defined as:

$$y = r_i + \gamma Q_i^{\pi'}(x', a_1', \ldots, a_N')|_{a_j'=\pi_j'(o_j)} \tag{13}$$

In this setup, the critic network leverages global information for learning, while the actor-network relies solely on local observation information. The cooperative policy framework facilitates agents in perceiving the state of other agents as an integral part of the environmental state, rather than treating them as distinct entities. By incorporating this state information, agents are empowered to update their own policies through the critic. In the context where full awareness of all agents' actions is present, and despite the persistent implementation of continuous policy updates, the environment itself remains stable and unchanged. Such a condition duly fulfills the requisite for policy gradient updates, thereby facilitating the generation of continuous actions. This approach facilitates achieving more accurate positioning in manipulator tasks and allows for motion planning in a 3D environment.

Through the training process, MAPPO can assign importance weights to each agent and efficiently learn decentralized policies that lead to effective coordination and cooperation among agents. We found this mechanism reflects a formal consistency with the priority coefficients of the inertia matrix M in the Riemannian motion policy. While RMPflow achieves policy synthesis by employing pullback operations along with priority coefficients, our approach enables end-to-end policy generation at the local agent level and

subsequently integrates these policies into a multi-agent collaborative policy through the use of CTDE methods.

E. REWARD FUNCTION

In the field of reinforcement learning and robotics, a common approach involves developing a comprehensive reward function that incorporates expert knowledge. This iterative process requires continuous interaction with the environment to finely tune parameters, ultimately leading to the achievement of an optimal reward function. Conversely, a simplistic sparse reward function relies on a straightforward sensor-based evaluation of task completion. However, utilizing such a reward function exposes the agent to uniformly negative rewards until a successful outcome is achieved, hindering the acquisition of valuable information necessary for policy improvement. To address this limitation, our study adopts a sparse reward function and introduces a long-term reward mechanism within the framework of multi-agent collaborative training. This approach effectively strikes a balance between considering short-term and long-term rewards. The design of the reward function consists of three distinct components:

$$r(s,a) = (\delta_1 R_1 + \delta_2 R_2 + \ldots + \delta_6 R_6 + \gamma R_d + R_t) - \phi \tag{14}$$

In this equation, The total reward is composed of contributions from Local and global rewards, each of the 6 joints, denoted as R_1 to R_6, based on whether the joints reach their target positions. These goals are specified sequentially from the bottom joint up, with each joint's target determined by the motion output of the agent for the joint below. The weights δ_1 to δ_6 represent the relative importance of each joint's motion to the overall movement of the manipulator, this weight is not necessary, and it is helpful to improve training efficiency.

The distance reward R_d represents the reward obtained by the agent as the end effector approaches the target. Specifically, for each reduction of $0.1cm$ in the Euclidean distance to the target, the agent receives a reward of 1. This is a hyper-parameter tuning based on experience, in highly dynamic environments with larger $1cm$ or greater accuracy ranges, reward acquisition can be easy, but results in larger errors. Using lower accuracy would make reward accumulation slow for training. This design can encourage agents to pursue higher tracking accuracy.

The time reward R_t accounts for the reward obtained upon completing an action within a specified timeframe. If the current cycle time is shorter than the previous cycle time, the agent is rewarded with a value of 25. On the other hand, the punishment term ϕ is associated with collisions with obstacles. In the event of a manipulator collision, the agent incurs a penalty of 75. We set the penalty score very high because we hope that the agent can regard obstacle avoidance as the main task, and the architecture of multi-agent cooperation can also avoid reward attenuation, which leads the agent to give up the tracking task.

It's worth noting that, within the multi-agent environment, the term "obstacles" encompasses not solely the conventional narrow definition but extends to encompass all other agents present. This comprehensive approach offers the distinct advantage of effectively addressing the potential occurrence of self-collision scenarios involving the manipulator. The discount factor γ is set to 0.99 to encourage exploration and balance the trade-off between short-term and long-term considerations.

TABLE 2. One of the agents' environment domain randomization parameters.

Randomization Parameter	Dimension	Value Range	Units
Position (Target)	3	0∼60	cm
Position (Robot)	3	0∼5	cm
Position (Obstacle)	3	0∼40	cm
Linear Velocity (Obstacle)	3	0.01∼0.3	m/s
Angular Velocity (Obstacle)	3	0∼0.261	rad/s
Mass (Robot Link)	1	4∼20	kg
Joint Static Ratio Errors	1	±0.03	None
Joint Dynamic Ratio Errors	1	±0.03	None
Joint Positions	1	0∼2π	rad
Joint Velocities	1	0∼2π	rad/s
Joint Torque Errors	1	±15	Nm
Link length	1	10∼75	cm

F. DOMAIN RANDOMIZATION

To enable agents to exhibit robustness to changes in the environment and generalize smoothly to the real world, we employ Domain Randomization to mitigate the reality gap issue in Reinforcement Learning, which is a widely used and proven method for solving Sim2Real problem [45]. Specifically, we train the agent in a simulated environment with randomized parameters designed to cover the distribution of environments that may be encountered in the real world.

To promote generalization of motion generation in highly dynamic environments, we introduce randomization of the position and motion parameters of obstacles and tracked targets. Moreover, to enable generalization across a wider range of manipulator models, we also randomize the physical parameters of the manipulator itself. Since the minimal control unit of our agent is a two-link model, randomization of these parameters does not dramatically alter the redundant action space, facilitating relatively straightforward training. The collective parameter settings can be referred to the TABLE 2.

The physical parameter ranges were selected with reference to the technical manuals of a variety of popular manipulators, view to cover mainstream situations. It is worth noting that the randomization parameter of joint positions and joint velocities is only used for pre-training, and the deployment is limited according to the parameters provided

by the model. The randomization of the link length parameter allows combining manipulators of different sizes in a multi-agent environment, which makes the policy robust to different models of manipulators.

IV. EXPERIMENT

Through the proposed framework, we simplify the design process of Riemannian motion policy, but the effect of this method still needs to be verified. The experimental part aims to verify the following three questions:

(1) Whether our framework allows local joints and global manipulators to effectively cope with dynamic tracking and obstacle avoidance tasks and is generalized?

(2) How harsh environment can the reaction movement generated by our method cope with?

(3) Can our approach improve the training efficiency when using reinforcement learning algorithms for manipulator tasks?

We use 6 common official URDF manipulator model for conducting simulation experiments in our study, the model is dynamically corrected. Each joint in this model is capable of receiving control signals independently. The detection of obstacles comes from the simulator, a collision detection mesh around the robot model with a warning distance of 15cm for collisions. To assess the efficacy of the proposed approach, we conducted three distinct experiments. Firstly, we implemented the suggested framework to pre-train an agent within a single joint environment. This experiment allowed us to examine the benefits offered by the proposed method in terms of training speed, while also evaluating its effectiveness in accomplishing a task involving dynamic target tracking and obstacle avoidance. Following the pre-training phase, we deployed the agent to each joint of the robot and trained it using Multi-agent Reinforcement Learning in a collaborative task scenario. Subsequently, we evaluated the agent's performance in the tracking-obstacle avoidance task within an environment containing multiple obstacles. Lastly, we aimed to assess the reactive action capabilities of the proposed method by examining the success rate of the task under varying obstacle speeds in dynamic obstacle environments.

To facilitate our Training and simulation, we utilized Omniverse Isaac Gym [8], an advanced tensor-based simulator specifically developed by Nvidia, different from the traditional RL simulation environment [20]. The utilization of Omniverse Isaac Gym offers the distinct advantage of seamlessly converting arrays into tensors through a tensor wrapper. This feature expedites the process of reinforcement learning training by allowing the execution of numerous parallel environments on a single GPU.

A. PRE-TRAINING RESULTS ON THE META-BEHAVIORAL LAYER

In the test environment of the meta-behavioral layer, a scenario comprising a simplified manipulator with a two-link mechanism, a moving target ball (colored red), an obstacle

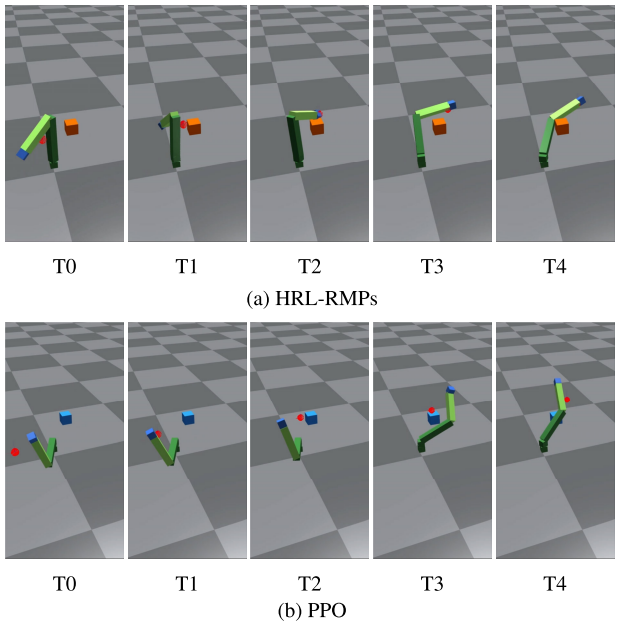


FIGURE 2. The figures show the results of pre-training an agent in a tracking-obstacle avoidance task, (a) using hierarchical reinforcement learning combined with RMP (HRL-RMPs), and (b) using standardized reinforcement learning only. After the same training time, HRL-RMPs can stably track the target (red dots) while avoiding obstacles in the path (squares), while RL can also avoid squares though, with a large error from tracking the target.

strategically positioned along the trajectory of the target ball, and importantly, the target ball itself lacked a collision volume. In each experiment, the target ball exhibited uniform linear motion in a random direction. The objective entailed the manipulator's end effector accurately tracking the target ball's position while actively avoiding any encountered obstacles within the tracking path. The agent's awareness of the obstacle locations was limited, as we solely relied on a collision detection network integrated into the manipulator model. This setup resulted in unexpected collision scenarios during the tracking process, necessitating the algorithm to promptly react and adapt to circumvent the obstacles effectively.

In our experiments, we employed two reinforcement learning algorithms, namely Proximal Policy Optimization (PPO) and a hierarchical structure-based algorithm that combines Reinforcement Learning with Riemannian Motion Policies (HRL-RMPs). The task setting chosen for evaluation was not particularly challenging for contemporary deep reinforcement learning algorithms. Notably, with sufficient training, PPO demonstrated relatively satisfactory performance in achieving desirable results. However, when considering the outcomes observed within a limited number of training iterations, our proposed algorithm showcased several distinct advantages.

In this section, we present the results of a representative experiment, showcasing the remarkable performance of our method in highly stable tracking trajectories with minimal errors, even after a relatively short training period. On the

TABLE 3. Obstacle avoidance success rate and tracking errors of pre-training environment.

	Success Rate	Avg. Tracking Error
HRL-RMPs (5K)	83.4%	22.6 ± 12.1
HRL-RMPs (10K)	95.6%	14.3 ± 5.4
HRL-RMPs (15K)	99.6%	6.3 ± 0.6
PPO (5K)	45.5%	172.6 ± 44.2
PPO (10K)	73.7%	76.2 ± 26.2
PPO (15K)	87.1%	44.8 ± 14.6

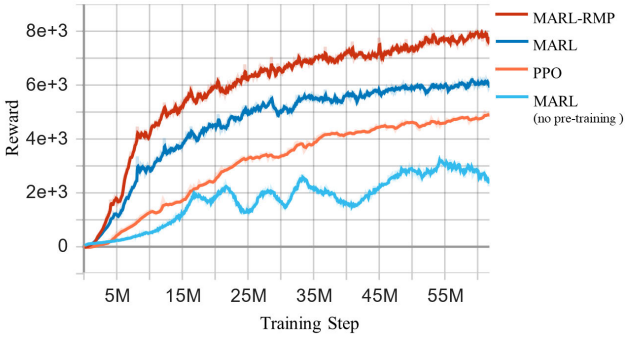


FIGURE 3. With the help of our learning framework, The training speed of the multi-agent policy is much faster than that of the single-agent policy. The main reason for the difference in reported scores stems from the success rate of the tasks. Our method was able to complete the task in more situations, which in turn allowed us to earn more points, while other methods failed in some of the more difficult situations resulting in the reward not being able to improve further.

other hand, the PPO algorithm is still in its initial training stages, and the observed motion reflects its early learning attempts to approach and track the target position, which falls short of the desired accuracy. In terms of obstacle avoidance, our method swiftly navigates around obstacles, resulting in only a slight deviation from the original path, after which it promptly resumes tracking the target trajectory. In contrast, the PPO algorithm also learns an obstacle avoidance policy but tends to take unnecessarily longer and less efficient avoidance paths. The outcomes of this experiment emphasize a significant improvement in both the efficiency and effectiveness of training through our approach, results are shown in FIGURE 2. We also compared the effect of different training steps on success rate and tracking accuracy. The data show that our method achieves a high success rate and high tracking accuracy with less training. Detailed results can be found in TABLE 4.

Following a brief comparison of the two algorithms, it becomes evident that RMP provides exceptional reactive motion capabilities within the underlying actions, effectively equipping the agent with valuable prior knowledge. Consequently, the agent's actions can be trained without starting from scratch, leading to notable improvements in both movement quality and training speed. Moreover, the hierarchical structure of our approach contributes to a more manageable and efficient action space, particularly when

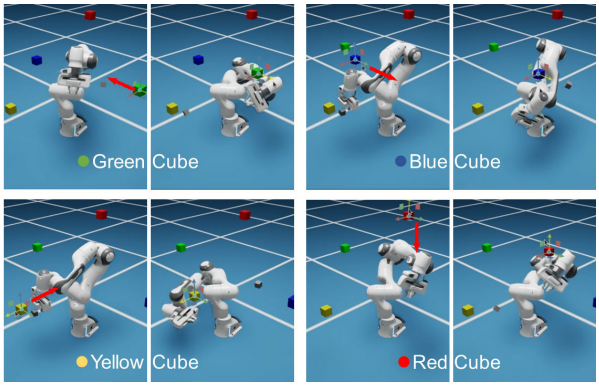


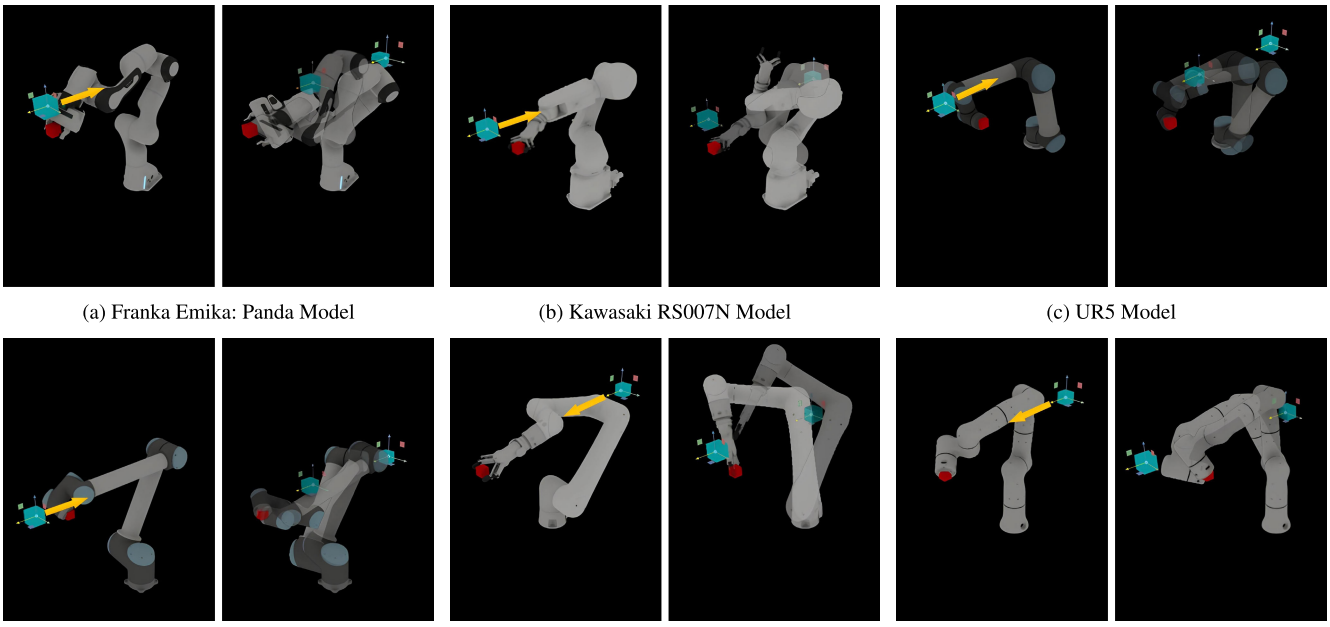
FIGURE 4. This image shows our approach to allow the manipulator to avoid dynamic obstacles from all directions and tests the motion of the agent on different joints. The red arrows in the figure mark the direction of the obstacle movement, while the next image shows the successful obstacle avoidance action.

compared to the direct control of joint torque. Collectively, these findings validate our hypothesis that combining hierarchical RL with RMP yields superior movement quality and accelerated training speed, thus providing preliminary evidence supporting the effectiveness of our approach. It is worth noting that pre-training is crucial to our method. Pre-training either way helps the results, but not doing it can make subsequent training of complex strategies extremely unstable. This difference will be shown in detail in the next experiment.

B. REACHING AND OBSTACLE AVOIDANCE

After successfully completing the pre-training and experiments involving a single intelligent body, we proceeded to investigate more complex policies. Specifically, we deployed the pre-trained agents to each joint of a high-dimensional manipulator with 7 DoF, forming a multi-agent collaborative system. This system was then trained using MAPPO. The experiment setup encompasses a dynamic environment: a full manipulator, a mobile target, and an obstacle. The initial motion direction of the obstacle is deliberately randomized to introduce variability. The primary objective for the manipulator is to effectively track the dynamic target while successfully avoiding collision with the obstacle. To enhance the robustness of the system, both the moving target and the obstacle possess randomly generated initial states within their respective domains.

We conduct a comparative analysis of the training speeds between four approaches: Proximal Policy Optimization (PPO), Multi-Agent Reinforcement Learning (MARL-PPO) with PPO pre-training, and MARL with Riemannian Motion Policies pre-training(MARL-RMP), and MARL without pre-training. The findings unequivocally demonstrate the superior efficiency of our proposed method. Specifically, when utilizing the MARL training framework with PPO, necessitates 2.5 times more training steps in comparison to our method. Moreover, the PPO algorithm exhibits a requirement of 4.5 times more training steps when compared to our approach. Notably, our method attains the highest



(a) Franka Emika: Panda Model (b) Kawasaki RS007N Model (c) UR5 Model

(d) UR10 Model (e) Denso Cobotta Pro 1300 Model (f) Flexiv: Rizon 4

FIGURE 5. The images show the effect of MARL-RMP on a variety of manipulators to test robustness. The manipulator needs to approach the target (red) and avoid moving obstacles (blue). Each set of images contains two, the left side shows the initial pose and the yellow arrow indicates the direction of movement of the obstacle. The right side shows the critical 2 frames during the movement of the obstacle and the manipulator, with the semi-transparent part earlier and the opaque one later.

final payoff scores, affirming its enhanced performance. In contrast, MARL without pre-training fails to produce stable training results, which is reflected in the non-convergence of the reward function. The reason for this is that using MARL directly in this task makes the exploration space very large, and we use a sparse reward function, which makes training difficult, this also proves the necessity of the pre-training step. Comprehensive results are visually depicted in FIGURE 3 for reference.

During testing, the environment comprised a manipulator, surrounded by four obstacles represented by red, green, yellow, and blue cubes. Additionally, a target cube, colored gray, moved in a uniform linear motion. This task required the manipulator to accurately track the moving target while simultaneously avoiding the dynamically moving obstacles. The increased dimensionality of both the obstacles and the manipulator movements introduced an additional layer of complexity to the task. It is worth mentioning that we allow artificially set the trajectory of the obstacle movement. This deliberate arrangement aimed to test the local motion policies of each joint within the manipulator.

In our study, we present several scenarios where obstacles introduce interference during the motion of the manipulator. These scenarios encompass various challenging situations, such as obstacles approaching the joints of the manipulator, obstacles traversing the working area, repeated interference with the tracking path, and the presence of multiple obstacles surrounding the manipulator. The experimental results highlight the efficiency and intelligence exhibited by our approach. Specifically, each joint of the manipulator effectively avoids obstacles approaching from multiple directions, demonstrating a robust obstacle avoidance capability. When an obstacle crosses the working area, the multi-agent collaboration policy enables the entire manipulator to avoid collisions and maintain a safe posture. In instances where obstacles interfere with the tracking path, our policy leverages the knowledge acquired during pre-training, allowing the manipulator to take a smaller additional path to bypass the obstacle while seamlessly continuing to track the target, results are shown in FIGURE 4.

Moreover, in scenarios involving multiple surrounding obstacles, which can potentially result in unsolvable motion paths, distinguishes itself from single-agent reinforcement learning methods. Instead of persistently attempting to navigate through the obstacles, which may lead to unnecessary jolts and dangerous movements, our policy learns to pause the motion and wait for the environment to change, exhibiting a more cautious and adaptive behavior. These experimental findings demonstrate the effectiveness of our approach in handling a wide range of obstacle interference scenarios, showcasing efficient and intelligent motion planning and execution.

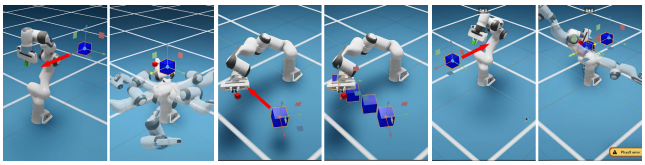
To evaluate the robustness of our method to variations in manipulator models, we leverage domain randomization to enable adaptation to differing physical parameters, including form factor differences. We selected 6 diverse manipulator models in widespread use across scenarios. Readers are advised to consult the technical manuals of the individual manipulators for detailed physical parameters. Mimicking the previous experimental setup, the manipulator must still

approach a target, while an obstacle crosses the working area from the front, interfering with multi-joint motion. The manipulator must avoid the obstacle while maintaining target tracking. As depicted in FIGURE 5, the results demonstrate the capability of our method to generalize effectively across manipulator models. Besides, our method possesses advantages in applying the randomization of the connecting rod length parameter. Variation in robot size changes the environment drastically, leading to a larger state space where DRL suffers from the catastrophic forgetting problem. However, in the MARL framework, modeling multi-agent systems based on cooperation can simplify the state dimension of the redundant manipulator, the local state change of each agent remains within acceptable limits and use centralized training to maintain collaboration. Therefore, catastrophic forgetting does not occur when applying link length randomization. This is the reason why our method in Figure 3 has significant advantages.

In addition to our proposed approach, we conducted a comparative analysis involving traditional motion planning using the Rapidly-Exploring Random Tree (RRT) method [1], a single-agent reinforcement learning algorithm (PPO) [5], and RMP flow in the presence of dynamic obstacles that approach the manipulator and traverse its working area. Results indicate that the RRT algorithm struggles to effectively cope with the highly dynamic environment presented in our experiments. Similarly, the single-agent reinforcement learning algorithm, PPO, encounters difficulties in certain cases. The direct control of high-dimensional movements by the agent, coupled with the random nature of neural network outputs, leads to instances where the agent outputs excessive torque. This excessive torque can result in damage to the manipulator. Regarding RMP flow, while the algorithm successfully avoids the obstacles, it does not converge back to a stable state and, instead, falls into a singular pose. results are shown in FIGURE 6. Quantitative analyses will be performed together with the next experiments. These comparative findings highlight the advantages of our proposed approach, which demonstrates efficient and intelligent motion planning and execution in the presence of dynamic obstacles, overcoming limitations observed in traditional motion planning methods, single-agent reinforcement learning algorithms, and RMPflow.

C. REACTIVE MOTION

To further evaluate the reactive motion capabilities of our proposed method, we conducted additional experiments in dynamic environments. In these experiments, the environment consisted of a manipulator tasked with tracking a moving target (gray cube). We introduced a line of dynamic obstacles that repeatedly crossed the working area of the manipulator at varying speeds, enabling us to measure the success rate of the manipulator's ability to avoid these obstacles. In another experiment, we scattered multiple obstacles randomly throughout the environment,



(a) RMPflow (b) RRT (c) PPO

FIGURE 6. The figures show some typical failures of RMPflow, RRT, and PPO when an obstacle crosses the working area of the manipulator (in the direction of the red arrow), Figure (a) shows that RMPflow most certainly avoids the obstacle but the manipulator fails to return to its normal attitude. Figure (b) shows that when the obstacle is already close to the manipulator but the RRT does not make an evasive maneuver. Figure (c) shows that the reinforcement learning algorithm PPO enters into a singular pose and fails to recover when avoiding an obstacle.

TABLE 4. Comparative simulation experiments for multi-obstacles avoidance tasks with different velocity.

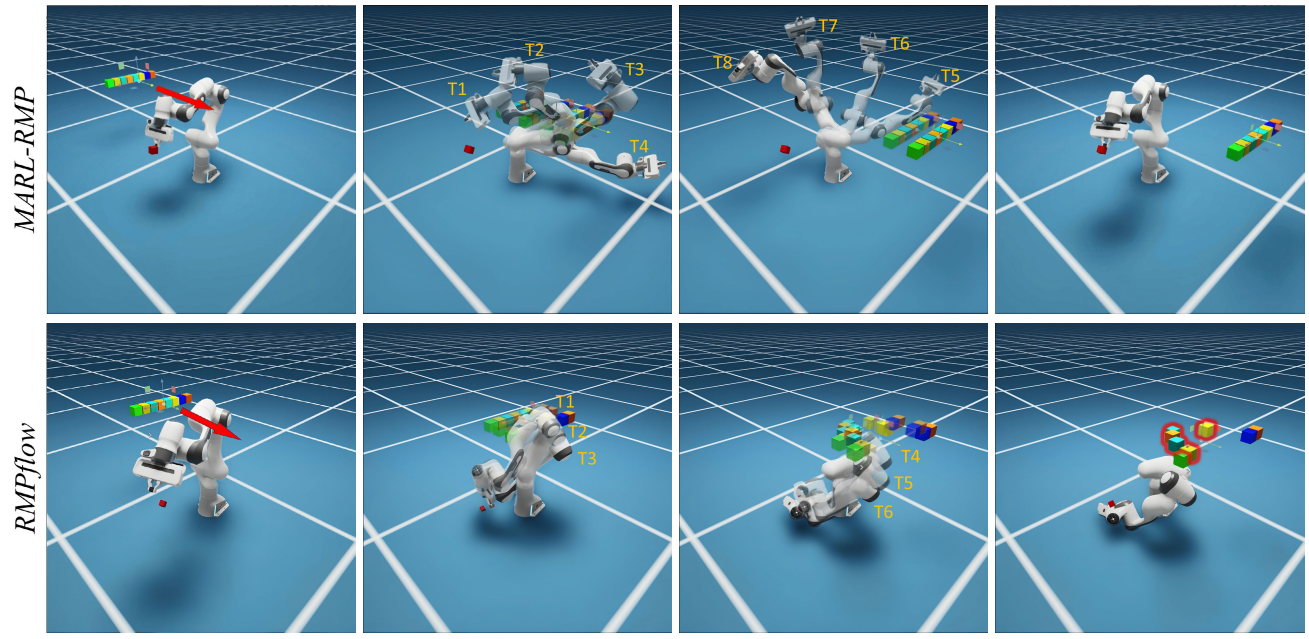
	0.2	0.6	1.2	1.8	2.8
PPO	90.4%	83.6%	*Failed*	*Failed*	*Failed*
MAPPO	95.2%	92.4%	52.2%	*Failed*	*Failed*
RMPflow	100%	94.4%	82.4%	74.8%	65.3%
MARL-RMP	100%	100%	87.6%	85.5%	80.5%

with lateral and vertical obstacles placed at random distances ranging from 20cm to 50cm. The objective here was for the manipulator to swiftly adapt its pose to avoid these haphazardly placed obstacles.

These experiments allowed us to assess the reactive motion capabilities of our proposed method in response to dynamic obstacles. By measuring the success rate of obstacle avoidance in the first experiment and observing the manipulator's ability to quickly adjust its pose in the presence of random obstacles in the second experiment, we aimed to validate the effectiveness of our approach in reacting and responding to unforeseen challenges in dynamic environments.

In the comparison of PPO, MARL, and RMPflow in the conducted tests, several observations were made. Firstly, PPO demonstrated limitations in effectively responding to obstacle avoidance when the speed of the dynamic obstacles exceeded a certain threshold (greater than 1). This indicated a challenge in adapting to high-speed environments. On the other hand, RMPflow exhibited an effective obstacle avoidance policy across all tested speeds. However, its success rate noticeably decreased at higher speeds, indicating limitations in maintaining a high success rate under extreme dynamic conditions. In contrast, our proposed method demonstrated a relatively stable performance across different obstacle speeds. The success rate did not decrease significantly as compared to RMPflow, even at higher speeds. Our method maintained a high success rate even in a high-speed environment with a speed of 2.8. More results of obstacle avoidance tests with obstacles of different speeds can be found in TABLE 4.

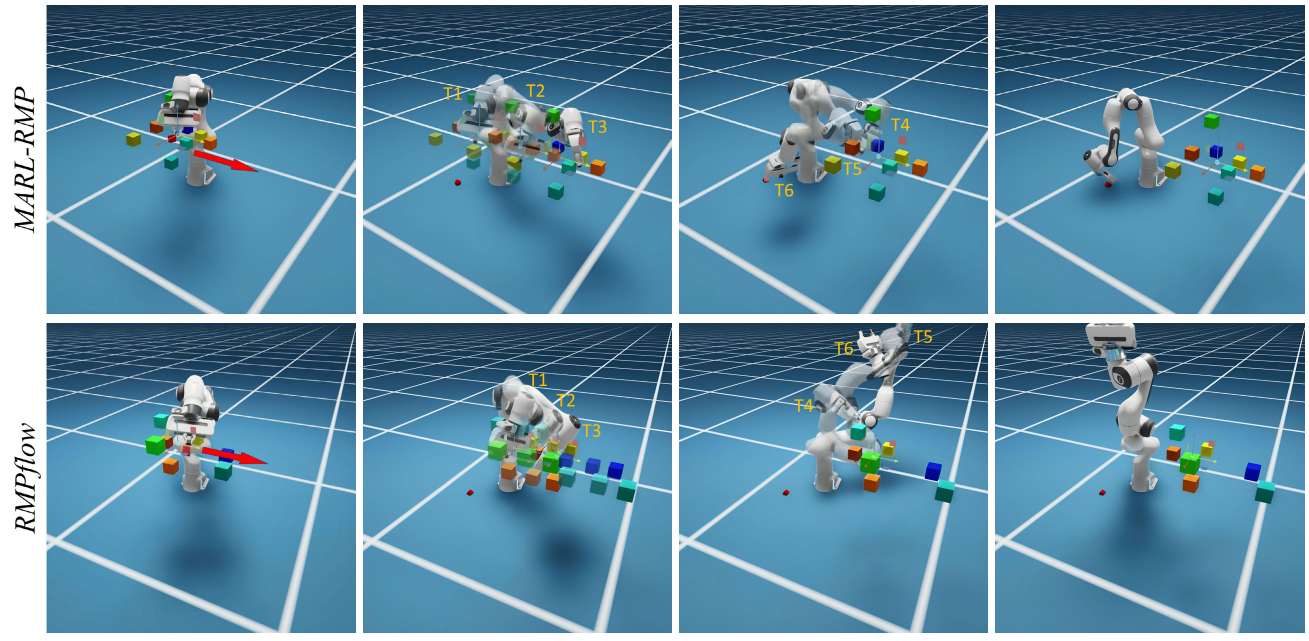
For the experiment involving avoiding cluttered objects, our method showcased greater stability in producing fast



(a) Initial Posture (b) Motion Process 1 (c) Motion Process 2 (d) Final Posture

Multi-Obstacle Avoidance Reactive Test for Multi-Joint

FIGURE 7. In the experiment, a sequence of obstacles is arranged linearly, traversing the operational region of the manipulator joint in alignment with the indicated red arrow. Upon encountering the manipulator, a colliding obstacle halts its motion, becoming stationary at the point of impact, while non-colliding obstacles continue their movement unhindered. The first line, utilizing our proposed method, showcases obstacles passing through the manipulator's working area while maintaining their original relative positions, indicative of a collision-free scenario. In contrast, the second line using RMPflow had a collision and the location of the collision is highlighted in red.



(a) Initial Posture (b) Motion Process 1 (c) Motion Process 2 (d) Final Posture

Multi-Obstacle Avoidance Reactive Test for End-effector

FIGURE 8. End-effectors often perform tasks in cluttered and dynamic environments, which this test simulates. End-effector is tracking the target (red cube), multiple obstacles passing through the End-effector working area, which also means that the position of the obstacles changes frequently and our reactive policy can be good at avoiding collisions and keep tracking the target. We show a typical failure situation in the second row. The baseline approach tries to avoid obstacles but ends up in a singularity that cannot be recovered.

responses. Conversely, RMPflow gradually struggled to adapt, leading to an inability to converge to a stable pose. These comparative results highlight the advantages of our proposed method in terms of maintaining stability and high

success rates in dynamic environments, especially at higher speeds, as well as its ability to respond quickly to challenging scenarios such as avoiding cluttered objects. For a typical result, refer to FIGURE 7 and FIGURE 8.

V. CONCLUSION

Our research focuses on the development and evaluation of a hierarchical RL and Multi-agent RL approach in conjunction with Reactive Motion Planning (RMP) for robotic motion planning and control, at the same time, we have greatly simplified the design of RMPs system. Through a comprehensive set of experiments and comparisons with traditional motion planning methods, single-agent RL algorithms, and RMP flow, we have demonstrated the effectiveness and advantages of our proposed approach across various dimensions. Our method showcases efficient and intelligent motion planning and execution capabilities, enabling precise target tracking and obstacle avoidance in both single-agent and multi-agent scenarios. Our approach yields stable tracking trajectories with minimal errors, facilitating swift and adaptive obstacle avoidance even in complex environments with dynamic obstacles and cluttered surroundings. Notably, our research emphasizes the superior training efficiency exhibited by our approach. This can be attributed to the hierarchical structure of our RL framework and the incorporation of RMP, which provides prior knowledge and simplifies the action space. In summary, our research presents a robust and efficient framework for reactive motion planning and control. It underscores the potential of hierarchical RL in combination with RMP to enhance both movement quality and training speed in robotic systems.

This research still has some limitations, we only explored the effect of the framework on manipulators, but should also have the ability to be applied to more types of robots, such as quadrupedal or humanoid robots, which are composed of joints and linkages, but the case of more nonlinear motion needs to be considered. We expect to validate more robotic platforms. In addition, while the reactive motion planning approach proposed in this work shows promise in simulation, several challenges remain in translating these results into real-world robotic systems [45]. First, reliable sensing of complex, dynamic environments is an open research problem, and the accuracy and responsiveness required may necessitate sophisticated multi-modal sensor systems. Occlusion effects pose additional difficulties as robots operate near obstacles or humans. Furthermore, while domain randomization helps account for physics discrepancies between simulation and reality, fully closing the ''reality gap'' requires physical validation not yet conducted here. We also need to add more safety policies to the RL training process to ensure that it does not pose a danger to the manipulator and personnel in real experiments [46]. Thus, future work should focus on perception system design and experimental deployment of the planning framework on physical manipulators. Through iterative research on perception, control, and planning, we aim to improve the method's applicability to real-world tasks.

REFERENCES

[1] D. Kalashnikov, A. Irpan, and P. Pastor, ''Scalable deep reinforcement learning for vision-based robotic manipulation,'' in *Proc. Conf. Robot Learn.*, Oct. 2018, pp. 651–673.

[2] J. J. Kuffner and S. M. LaValle, ''RRT-connect: An efficient approach to single-query path planning,'' in *Proc. ICRA Millennium Conf. IEEE Int. Conf. Robot. Automat. Symposia*, Apr. 2000, pp. 995–1001.

[3] D. Nguyen-Tuong and J. Peters, ''Model learning for robot control: A survey,'' *Cognit. Process.*, vol. 12, no. 4, pp. 319–340, Nov. 2011.

[4] L. Busoniu, B. De Schutter, and R. Babuska, ''Decentralized reinforcement learning control of a robotic manipulator,'' in *Proc. 9th Int. Conf. Control, Autom., Robot. Vis.*, 2006, pp. 1–6.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, ''Proximal policy optimization algorithms,'' 2017, *arXiv:1707.06347*.

[6] J. Peters and S. Schaal, ''Natural actor-critic,'' *Neurocomputing*, vol. 71, nos. 7–9, pp. 1180–1190, Mar. 2008.

[7] V. Kumar, D. Hoeller, B. Sundaralingam, J. Tremblay, and S. Birchfield, ''Joint space control via deep reinforcement learning,'' in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 3619–3626.

[8] J. Liang, V. Makoviychuk, and A. Handa, ''GPU-accelerated robotic simulation for distributed reinforcement learning,'' in *Proc. Conf. Robot Learn.*, Oct. 2018, pp. 270–282.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, ''Continuous control with deep reinforcement learning,'' 2015, *arXiv:1509.02971*.

[10] S. Li, J. K. Gupta, P. Morales, R. Allen, and M. J. Kochenderfer, ''Deep implicit coordination graphs for multi-agent reinforcement learning,'' 2020, *arXiv:2006.11438*.

[11] Y. Ansari, E. Falotico, Y. Mollard, B. Busch, M. Cianchetti, and C. Laschi, ''A multiagent reinforcement learning approach for inverse kinematics of high dimensional manipulators with precision positioning,'' in *Proc. 6th IEEE Int. Conf. Biomed. Robot. Biomechatronics (BioRob)*, Jun. 2016, pp. 457–463.

[12] B. Lucian, R. Babuska, and B. D. Schutter, ''Multi-agent reinforcement learning: A survey,'' in *Proc. 9th Int. Conf. Control, Autom., Robot. Vis.*, Dec. 2006, pp. 1–6.

[13] D. M. Stocco, ''A star search: Implications in controlling Steroidogenesis1,'' *Biol. Reproduction*, vol. 56, no. 2, pp. 328–336, Feb. 1997.

[14] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, ''The surprising effectiveness of PPO in cooperative, multi-agent games,'' 2021, *arXiv:2103.01955*.

[15] A. Perrusquía, W. Yu, and X. Li, ''Redundant robot control using multi agent reinforcement learning,'' in *Proc. IEEE 16th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2020, pp. 1650–1655.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[17] V. Pong, S. Gu, M. Dalal, and S. Levine, ''Temporal difference models: Model-free deep RL for model-based control,'' 2018, *arXiv:1802.09081*.

[18] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, ''A survey on deep reinforcement learning algorithms for robotic manipulation,'' *Sensors*, vol. 23, no. 7, p. 3762, Apr. 2023.

[19] R. Lowe, Y. Wu, Aviv Tamar, J. Harb, P. Abbeel, and I. Mordatch, ''Multi-agent actor-critic for mixed cooperative-competitive environments,'' in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, Dec. 2017, p. 6382 6393.

[20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, ''OpenAI gym,'' 2016, *arXiv:1606.01540*.

[21] M. Mukadam, ''Riemannian motion policy fusion through learnable Lyapunov function reshaping,'' in *Proc. Conf. Robot Learn.*, 2020, pp. 204–219.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, ''Playing Atari with deep reinforcement learning,'' 2013, *arXiv:1312.5602*.

[23] M. Botvinick and A. Weinstein, ''Model-based hierarchical reinforcement learning and human action control,'' *Phil. Trans. Roy. Soc. B, Biol. Sci.*, vol. 369, no. 1655, Nov. 2014, Art. no. 20130480.

[24] S. Gu, E. Holly, T. Lillicrap, and S. Levine, ''Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,'' in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 3389–3396.

[25] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," 2018, *arXiv:1801.02854*.
[26] F.-Y. Wang, J. J. Zhang, X. Zheng, X. Wang, Y. Yuan, X. Dai, J. Zhang, and L. Yang, "Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond," *IEEE/CAA J. Autom. Sinica*, vol. 3, no. 2, pp. 113–120, Apr. 2016.
[27] K. Arulkumaran, A. Cully, and J. Togelius, "AlphaStar: An evolutionary computation perspective," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2019, pp. 314–315.
[28] H. Cuayhuitl, S. Renals, O. Lemon, and H. Shimodaira, "Hierarchical reinforcement learning for spoken dialogue systems," in *Proc. 41st Annu. Meeting Assoc. Comput. Linguistics*, 2009, pp. 395–429.
[29] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2016.
[30] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, Nov. 2000.
[31] B. Beyret, A. Shafti, and A. A. Faisal, "Dot-to-dot: Explainable hierarchical reinforcement learning for robotic manipulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 5014–5019.
[32] K. Hansel, J. Urain, J. Peters, and G. Chalvatzaki, "Hierarchical policy blending as inference for reactive robot control," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, London, U.K., 2023, pp. 10181–10188, doi: 10.1109/ICRA48891.2023.10161374.
[33] M. Botvinick, S. Niv, and A. G. Barto, "Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective," *Trends Cognit. Sci.*, vol. 12, no. 10, pp. 442–447, Oct. 2008.
[34] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "RMPflow: A geometric framework for generation of multitask motion policies," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 3, pp. 968–987, Jul. 2021.
[35] A. Li, C.-A. Cheng, M. Asif Rana, M. Xie, K. Van Wyk, N. Ratliff, and B. Boots, "RMP2: A structured composable policy class for robot learning," 2021, *arXiv:2103.05922*.
[36] M. Shahrabi and M. Adibi, "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," in *Proc. IEEE Int. Conf. Autom. Comput. (ICAC)*, Sep. 2017, pp. 1–6.
[37] E. Aljalbout, J. Chen, K. Ritt, M. Ulmer, and S. Haddadin, "Learning vision-based reactive policies for obstacle avoidance," in *Proc. Conf. Robot Learn.*, Oct. 2021, pp. 2040–2054.
[38] M. Mattamala, N. Chebrolu, and M. Fallon, "An efficient locally reactive controller for safe navigation in visual teach and repeat missions," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 2353–2360, Apr. 2022.
[39] T. Shomrat, A. L. Turchetti-Maia, N. Stern-Mentch, J. A. Basil, and B. Hochner, "The vertical lobe of cephalopods: An attractive brain structure for understanding the evolution of advanced learning and memory systems," *J. Comparative Physiol. A*, vol. 201, pp. 947–956, Jun. 2015.
[40] T. Li, K. Srinivasan, M. Q. Meng, W. Yuan, and J. Bohg, "Learning hierarchical control for robust in-hand manipulation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 8855–8862.
[41] M. A. Rana, A. Li, D. Fox, S. Chernova, B. Boots, and N. Ratliff, "Towards coordinated robot motions: End-to-end learning of motion policies on transform trees," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 7792–7799.
[42] R. Rana, Y. Li, A. Gupta, D. Fox, and L. Pinto, "Learning reactive motion policies in multiple task spaces from human demonstrations," in *Proc. Conf. Robot Learn.*, 2020, pp. 1457–1468.
[43] Y. Wang and R. Sagawa, "Manipulator motion planning via centralized training and decentralized execution multi-agent reinforcement learning," in *Proc. Int. Conf. Adv. Robot. Mechatronics (ICARM)*, Jul. 2022, pp. 812–817.
[44] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers Robot. AI*, vol. 9, Apr. 2022, Art. no. 799893.
[45] A. Bonci, P. D. Cen Cheng, M. Indri, G. Nabissi, and F. Sibona, "Human-robot perception in industrial environments: A survey," *Sensors*, vol. 21, no. 5, p. 1571, Feb. 2021.
[46] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang, and A. Knoll, "A review of safe reinforcement learning: Methods, theory and applications," 2022, *arXiv:2205.10330*.

YULIU WANG received the B.S. degree from the Wuhan Institute of Technology, in 2015, and the M.S. degree from the University of Tsukuba, in 2021, where he is currently pursuing the Ph.D. degree with the Joint Graduate School of Intelligent and Mechanical Interaction Systems. He was a Senior Technical Specialist with the Ongoing Reliability Testing Laboratory, Mobile Business Unit, Lenovo Group. He is a Research Assistant with the Computer Vision Research Group, Artificial Intelligence Research Center (AIRC-CVRT), National Institute of Advanced Industrial Science and Technology (AIST), Japan. His current research interests include robot motion planning, robot learning, and deep reinforcement learning.

RYUSUKE SAGAWA (Member, IEEE) received the B.E. degree in information science from Kyoto University, in 1998, and the M.E. degree in information engineering and the Ph.D. degree in information and communication engineering from The University of Tokyo, in 2000 and 2003, respectively. He was an Assistant Professor with the Institute of Scientific and Industrial Research, Osaka University, and moved to AIST, in 2010. He is currently an Associate Professor with the Cooperative Graduate School, Tokyo University of Agriculture and Technology, and the University of Tsukuba. He is also a Leader with the Computer Vision Research Team, Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Japan. His current research interests include computer vision, computer graphics, robotics, and human modeling (mainly geometrical modeling and visualization).

YUSUKE YOSHIYASU (Member, IEEE) received the Ph.D. degree from Keio University, Japan. He was a Visiting Scholar with the Leonidas Guibas Laboratory, Computer Science Department, Stanford University, from 2015 to 2016. He is currently a Senior Research Scientist with the National Institute of Advanced Industrial Science and Technology (AIST), Japan. He is a member of the Computer Vision Research Team, AI Research Center. He is an Adjunct Member of CNRS-AIST Joint Robotics Laboratory (JRL). His current research interests include shape analysis, computer vision, robot vision, and machine learning.