aerospace

MDPI

Article

Parallel Reconfiguration Planning for Modular Self-Reconfigurable Satellites via Multi-Agent Reinforcement Learning

Bo Wang *, Shilong Li, Jialin Yu, Dong Ye and Zhaowei Sun

Research Center of Satellite Technology, Harbin Institute of Technology, Harbin 150001, China; lsl01012001@163.com (S.L.); xucytltdrk3@163.com (J.Y.); yed@hit.edu.cn (D.Y.); sunzhaowei@hit.edu.cn (Z.S.)
* Correspondence: wangbo1028@hit.edu.cn

Abstract

Modular self-reconfigurable satellites can adapt to diverse space missions by dynamically rearranging their configurations. Enabling multiple modules to move simultaneously enhances reconfiguration efficiency but remains constrained by existing planning algorithms. This article introduces the Parallel Reconfiguration Planner (PRP), a learning-based approach that ensures safe, concurrent module movements. PRP leverages multi-agent reinforcement learning to generate reconfiguration policies while incorporating safety constraints modeled using Linear Temporal Logic (LTL). To improve learning efficiency, PRP integrates a guarded exploration mechanism based on these safety specifications. Experiments on 3D reconfiguration scenarios demonstrate that PRP achieves more efficient reconfiguration policies in scenarios that have more moving modules compared to the baseline method.

Keywords: modular self-reconfigurable satellite; parallel reconfiguration planning; multi-agent reinforcement learning; modular reconfigurable robotic system; reconfiguration planning; reinforcement learning

1. Introduction

The modular self-reconfigurable satellite (MSRS) is a type of modular self-reconfigurable robotic system [1] composed of multiple interconnected modules [2–4]. These modules serve different functions, including payload modules, structural modules, battery modules, and Guidance, Navigation, and Control (GNC) modules. These modules are able to move relative to one another, allowing the system to rearrange its shape, commonly referred to as the satellite's configuration [5,6]. By changing the configurations, the MSRS achieves greater flexibility, making it well-suited for diverse space missions [7–11]. The MSRS has adopted the pivoting cube model [12] (as seen in Figure 1), which belongs in the lattice architecture category. An example of this model is M-blocks [13,14], where the modules connect along the faces and edges using magnets and achieve locomotion through pivoting driven by internal flywheels. The connection–locomotion mechanism is both simple and effective. Each module shares a similar architecture to a cubic satellite [15] and employs a pollution-free actuation system, making it particularly suitable for space applications. Research on MSRSs spans several aspects (see Figure 2): hardware design [14,16,17], task-shape matching [18–20], reconfiguration planning [5,6,21–23], and reconfiguration control [24–26]. Our work focuses on reconfiguration planning, specifically, developing algorithms that

Check for updates

Academic Editors: Andrew W. H. Ip, Zhuming Bi and Kai Leung Yung

Received: 7 July 2026
Revised: 8 August 2026
Accepted: 8 August 2026
Published: 17 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

generate action command sequences to guide MSRSs from an initial configuration to a target configuration.

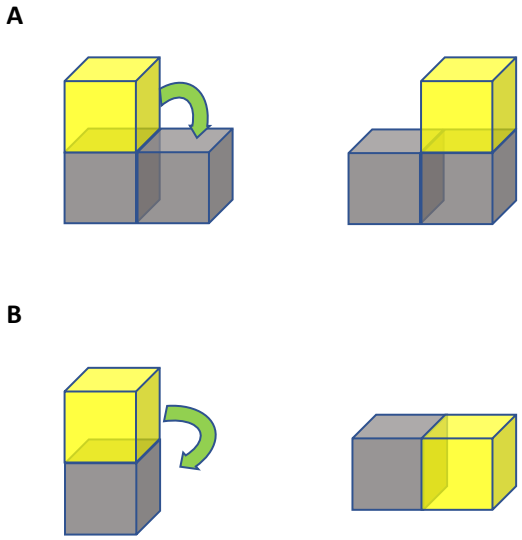


Figure 1. Illustration of pivoting cube model. Yellow cubes are moving. Grey cubes are static. (**A**) shows the yellow cube pivoting 90°. (**B**) shows the yellow cube pivoting 180°.

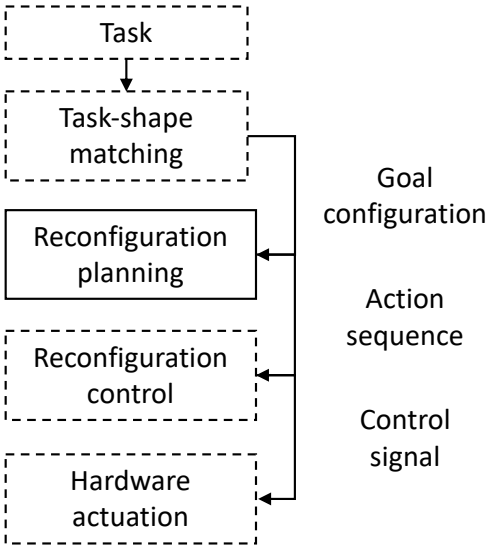


Figure 2. Overview of the modular self-reconfigurable satellite task. The solid-line box highlights the role of reconfiguration planning in the task.

Although the hardware of an MSRS is designed to allow multiple modules to move simultaneously, most existing reconfiguration planning algorithms [5,6,12,21,27–30] require the modules to move sequentially, one by one; therefore, the reconfiguration process is not efficient enough. Ref. [31] enables parallel movement; however, it solves swarm tasks instead of parallel reconfiguration tasks. In this article, we explore parallel reconfiguration planning, where all movable modules are considered capable of moving simultaneously during the planning process. One of the challenges in designing a parallel reconfiguration algorithm is that, when modules move together, collisions become more likely, and maintaining the integrity of the configuration connection becomes more difficult. To address this, we model the safety specifications for parallel reconfiguration using Linear Temporal Logic (LTL). Another challenge arises from the hierarchical nature of most existing algorithms, which typically first select a module and its target position, then plan a path from its initial position to the target using graph-based methods such as BFS [21] or A* [28–30]. This hierarchical

approach is not well-suited for parallel reconfiguration, as the planned path for a single module may be disrupted by the motion of other modules. To overcome this limitation, we propose a deep reinforcement learning-based algorithm that directly predicts the pivoting actions of each movable module, enabling concurrent motion while avoiding conflicts.

Deep reinforcement learning (DRL) has found numerous applications in motion planning and control [32–35]. However, the application of reinforcement learning to reconfiguration planning for an MSRS is still limited. Ref. [36] proposed a deep Q-learning-based algorithm for reconfiguration planning. Ref. [6] introduced a deep recurrent Q-learning-based algorithm for planning a single module's path in partially observable scenarios. Ref. [27] introduced a goal-orientated deep reinforcement learning algorithm. These approaches, however, focus on moving one module at a time. To enable parallel reconfiguration, the action space must be redefined, and potential action conflicts need to be addressed. Furthermore, to enhance learning efficiency, it is crucial to integrate a multi-agent framework into reinforcement learning algorithms. While numerous multi-agent reinforcement learning algorithms exist [37–41], most do not address the issue of safe exploration. Although there are safe exploration techniques, including those designed specifically for multi-agent systems [42], none are tailored for parallel reconfiguration planning. In this article, we introduce a novel multi-agent reinforcement learning algorithm, trained within a custom-built simulator designed for MSRS parallel reconfiguration. Our algorithm incorporates guarded exploration to minimize conflicts during the learning process. To the best of our knowledge, this is the first reinforcement learning-based approach specifically aimed at parallel reconfiguration planning for an MSRS.

Our contributions are highlighted as follows:

1. Formulating the parallel reconfiguration planning of an MSRS as a multi-agent Markov decision process, including the modeling of its state representation, action space, and reward function.
2. Modeling the single-agent and two-agent safety specifications with Linear Temporal Logic (LTL).
3. Proposing a multi-agent reinforcement learning-based algorithm named the Parallel Reconfiguration Planner (PRP) to solve parallel reconfiguration planning problem.
4. Introducing guarded exploration to enable safer exploration during learning, which leverages graph-based conflict modeling and an evolutionary conflict solver.
5. Evaluating PRP on 3D reconfiguration scenarios and demonstrating its ability to achieve parallel and efficient reconfiguration.

The meaning of abbreviations can be found in the Table 1.

Table 1. Table of abbreviations.

Abbreviation	Meaning
MSRS	Modular self-reconfigurable satellite
PRP	Parallel reconfiguration planner
LTL	Linear temporal logic
GNC	Guidance, Navigation, and Control
DRL	Deep reinforcement learning
MMDP	Multi-agent Markov decision process
ANN	Artificial neural network
VDN	Value-decomposed Q-learning
DQN	Deep Q-learning
MLP	Multi-layer perceptron
AM	Action mask
CS	Centralized shielding

2. Problem Formulation

2.1. Parallel Reconfiguration

Reconfiguration is the process of an MSRS changing its arrangement from an initial configuration c_0 to a goal configuration c_g. When multiple modules move simultaneously during this process, it is referred to as parallel reconfiguration. In an MSRS, some modules are designed to be movable, while others are fixed. The movable modules are also referred to as agents, as they assess their observation at the current time step and select an action for the next time step.

State: The state of an MSRS is its configuration c_t, where t demonstrates the time step. Several types of configuration representation will be introduced in Section 2.2. The relative position information can be propagated through connected modules; therefore, the entire configuration can be observed by traversing the connected structure [5]. Each agent has its own observation. We use $\mathbf{o}_t$ to represent joint observation and o_t^i to represent observation of agent i. Observation includes MSRS's configuration c_t and agent's position v_t^i. The states where the MSRS either reaches the goal configuration, fails due to unsafe actions, or terminates exploration are referred to as terminal states. We use o_f to denote the observation received by agents in a terminal state.

Actions: In an MSRS, each module is a cube. Agents move by pivoting (rotating) around the edges they share with other modules, following the rules of the pivoting cube model [12]. This model abstracts away inertia, actuation delays, force limits, and joint errors, leaving these factors to be addressed in reconfiguration control studies. The action of a single agent defines its movement, as described in [6]: with action a, its supporting neighbor index n, pivot edge L, and pivot angle θ will be

$$\begin{cases} n = a - \lfloor \frac{a}{6} \rfloor \cdot 6 \\ L = a - \lfloor \frac{a}{4} \rfloor \cdot 4 \\ \theta = 90^\circ [1 + (a - \lfloor \frac{a}{2} \rfloor \cdot 2)] \end{cases} \tag{1}$$

Here, the pivot edge $L \in \{1, 2, 3, 4\}$ is the edge around which the agent rotates, the supporting neighbor index $n \in \{1, 2, 3, 4, 5, 6\}$ is the index of module that provides the pivot edge, and the pivot angle $\theta \in \{90^\circ, 180^\circ\}$ is the angle by which the agent rotates. For parallel reconfiguration, an additional action $a = 0$ is included in the action space to indicate that the agent has chosen to remain stationary. At each time step, an agent's action is selected based on its individual policy π^i. The joint policy $\boldsymbol{\pi}$, derived from the set of individual policies, maps the global state to a distribution over joint action $\mathbf{a}$.

For brevity, we denote the configuration at the next time step as c', the observation of agent i as $o^{i\prime}$, the joint action as $\mathbf{a}'$, and the individual action of agent i as $a^{i\prime}$.

Reward: After executing a joint action $\mathbf{a}_t$, the MSRS transitions to a new configuration c_{t+1}. If c_{t+1} is goal configuration c_g, all agents receive reward $r_{t+1} = r_g$. If $\mathbf{a}_t$ is not safe, all agents receive penalty $r_{t+1} = r_p$. Otherwise, all agents receive an intrinsic reward $r_{t+1} = -d(c_{t+1}, c_g) r_i$. Here, r_i is a positive constant, and $d(\cdot)$ represents the distance between two configurations.

In this article, we use the optimal assignment metric defined in [43] as $d(\cdot)$. This metric defines the distance between two configurations as the optimal assignment of each element in the first configuration to an element in the second configuration, such that the sum of the Manhattan distances for the assignment is minimized. The assignment can be computed using the Hungarian algorithm, and the resulting sum of distances is the optimal assignment metric. In parallel reconfiguration, all agents share the same reward.

The above setup of parallel reconfiguration is a special instance of what is known in the RL community as a multi-agent Markov decision process (MMDP). In an MMDP, all

agents are cooperative and aim to maximize the cumulative global reward. For any joint policy $\boldsymbol{\pi}$, the action-value function $Q^{\boldsymbol{\pi}}(c, \mathbf{a})$ represents the expected return starting from configuration c, taking joint action $\mathbf{a}$, and subsequently following the joint policy $\boldsymbol{\pi}$. It is defined as

$$Q^{\boldsymbol{\pi}}(c, \mathbf{a}) = \mathbb{E}_{\boldsymbol{\pi}, Pr}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid c_0 = c, \mathbf{a}_0 = \mathbf{a}\right] \tag{2}$$

where $\mathbb{E}_{\boldsymbol{\pi}, Pr}(\cdot)$ denotes the expectation over trajectories generated by following $\boldsymbol{\pi}$ under the transition dynamics Pr, and γ is the discount factor. The action-value function for the optimal joint policy $\boldsymbol{\pi}^*$ is defined as $Q^*(c, \mathbf{a}) = \sup_{\boldsymbol{\pi}} Q^{\boldsymbol{\pi}}(c, \mathbf{a})$. The goal of parallel reconfiguration planning is to find this optimal joint policy $\boldsymbol{\pi}^*$.

2.2. Configuration Representation

This article introduces an artificial neural network (ANN) for approximating the action-value function, known as a Q-network, along with several safety specifications for safe exploration. We propose a matrix representation of configurations as input for the ANN and use graph and set representations to help define these safety specifications.

2.2.1. Matrix Representation

Let v_i represent the position vector of module i. For simplicity, v_i is also used to donate module i itself. The configuration of an MSRS with M modules can be represented as an $M \times 3$ matrix, V. Each row of V corresponds to the position vector of a module.

2.2.2. Set Representation

A set $\mathcal{V} = \{v_1, v_2, \ldots, v_M\}$ can represent the configuration, where v_i (for $i = 1, 2, \ldots, M$) denotes the i-th module of the MSRS. The intersection of two sets, $\mathcal{V}^a \cap \mathcal{V}^b$, represents the modules that occupy the same positions. Sets can also represent positions, such as the obstacle position set $\mathcal{V}^o$. The intersection $\mathcal{V} \cap \mathcal{V}^o$ indicates the modules located at obstacle positions.

2.2.3. Graph Representation

An undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ can represent a configuration, where the vertex set $\mathcal{V}$ represents the modules, and the edge set $\mathcal{E}$ represents the connections between pairs of modules. In an MSRS, two modules joined by a face are considered connected. The graph $\mathcal{G}$ is considered connected if, for every pair of vertices v_i and v_j in $\mathcal{V}$, there exists a path (a sequence of edges) that starts at v_i and ends at v_j.

3. Safety Specification

The MSRS breaks down when modules collide or when the configuration loses connectivity [12]. The motion constraint that prevents these two situations is referred to as the safety specification. If joint actions violate the safety specification, the parallel reconfiguration process fails, and the corresponding MMDP reaches its terminal state. We use Linear Temporal Logic (LTL) [44] to express the safety specification. In addition to propositional logic operators, LTL includes temporal operators such as $\Box$ (always) and $\bigcirc$ (next). The LTL formula for the safety specification of parallel reconfiguration is $\Box\neg((\text{two modules collide}) \vee (\mathcal{G} \text{ is disconnected}))$. Although some collisions can be avoided through carefully designed control policies, this work focuses on reconfiguration planning. Accordingly, the proposed safety specifications are designed to guarantee safety regardless of the underlying control policy.

To provide a more detailed formulation, the safety specification is divided into single-agent and multi-agent safety specifications. The single-agent safety specification ensures that each agent can operate independently in a safe manner, assuming other agents remain

static, represented as φ_s (see Section 3.2). In the context of parallel reconfiguration, a conflict occurs when the actions of multiple agents individually satisfy the single-agent safety specification but their simultaneous execution results in unsafe conditions, represented as φ_c. The LTL formula for the safety specification of parallel reconfiguration is $\square\neg\varphi_c \wedge \varphi_s$. The multi-agent safety specification ensures that joint actions avoid conflicts.

In this article, we formulate the two-agent safety specification (see Section 3.3), which addresses conflicts between any two agents. Some safety violations arise when more than two agents move simultaneously, potentially disrupting the connectivity of the configuration. Rather than explicitly modeling these scenarios, this paper directly verifies whether the new configuration remains connected. In cases where connectivity is lost, agents receive a negative global reward, encouraging them to learn to avoid such situations.

3.1. Influencing Area

The first step is to define the influencing area of each agent's movement. For each agent, modules connected by face are the neighbors. Each module can have at most six neighbors. We define the neighbor matrix

$$Ne \doteq \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

With this matrix, the position of the supporting neighbor can be represented as

$$v_n = v_i + Ne[n] \tag{3}$$

where $Ne[n]$ is the nth row of the matrix Ne. The supporting neighbor is the module that provides a pivot for the moving module to rotate.

To formulate safety specifications, the influencing area is defined, which is shown in Figure 3. The position vectors of positions in the influencing area can be calculated as

$$\begin{cases} v_1 = v_i + Ne[7-n] \\ v_2 = v_1 + Ne^-[L] \\ v_3 = v_i + 2 \cdot Ne^-[L] \\ v_4 = v_n + 2 \cdot Ne^-[L] \\ v_5 = v_i + Ne^-[L] \\ v_6 = v_n + Ne^-[L] \end{cases} \tag{4}$$

where subscript is the mark of the position, which can be found in Figure 3, and Ne^- is a new matrix generated by removing the jth line and the $(7-n)$th line of the neighbor matrix Ne.

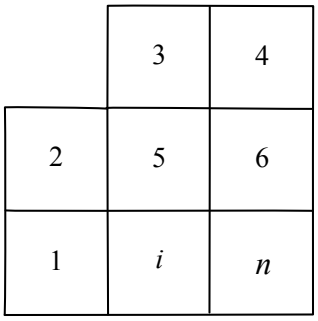


Figure 3. 2D projection of the influencing area. The calculation of each position's position vector is shown in Equation (4).

Positions in influencing area can be divided into three types:

(1) Obstacle: While pivoting, the agent will sweep through obstacle positions. Thus, modules in these positions will be the obstacles of locomotion. If $\theta = 90^\circ$, the obstacle set is $\mathcal{V}^o = \{v_1, v_2, v_5\}$. If $\theta = 180^\circ$, the obstacle set is $\mathcal{V}^o = \{v_1, v_2, v_3, v_4, v_5, v_6\}$.
(2) New neighbor: Modules connected with the agent by face. After pivoting, the agent will form new magnetic connections with modules at the new neighbors set, denoted as set $\mathcal{N}$. If $\theta = 90^\circ$, $\mathcal{V}^n = \{v_3, v_6\}$. If $\theta = 180^\circ$, $\mathcal{V}^n = \{v_n\}$.
(3) Supporting neighbor: The neighbor provides a pivot for the moving module to rotate, denoted as v_n.

3.2. Single-Agent Safety Specification

Each agent's pivoting involves the requirement of having one supporting neighbor, indicating a stable structural foundation during movement. Furthermore, there is an absence of obstacles, avoiding collision risks. Additionally, the module must establish connection with at least one new neighbor after pivoting, ensuring the maintenance of connectivity. This is described as follows:

Safety Specification 1. *(Individual movement) For agent v_i, its supporting neighbor denotes v^n, the obstacle position set denotes $\mathcal{O}$, and neighbor position set denotes $\mathcal{N}$; it follows* $\square((\mathcal{V}^o \cap \mathcal{V} = \emptyset) \wedge (v_n \in \mathcal{V}) \wedge \bigcirc\neg(\mathcal{V}^n \cap \mathcal{V} = \emptyset) \wedge ((\mathcal{G} - v_i)$ *is connected*$))$.

3.3. Two-Agent Safety Specification

3.3.1. Collision Free

To ensure collision-free parallel reconfiguration, a strict rule is applied to every pair of agents. The obstacle areas associated with these agents must not overlap at any point during the reconfiguration process. This constraint guarantees that the trajectories of any two agents do not intersect, thereby preventing collisions and minimizing the risk of damage during reconfiguration. This is described as

Safety Specification 2. *(Collision free) For two agents, v_i and v_j, their obstacle position sets are $\mathcal{V}^{o_i}$ and $\mathcal{V}^{o_j}$, respectively. Their actions follow* $\square((\mathcal{V}^{o_i} \cap \mathcal{V}^{o_j} \neq \emptyset) \rightarrow ((a^i = 0) \vee (a^j = 0)))$.

This means, for every two moving modules, their obstacle positions cannot overlap. Figure 4A is an example of violating the specification 2.

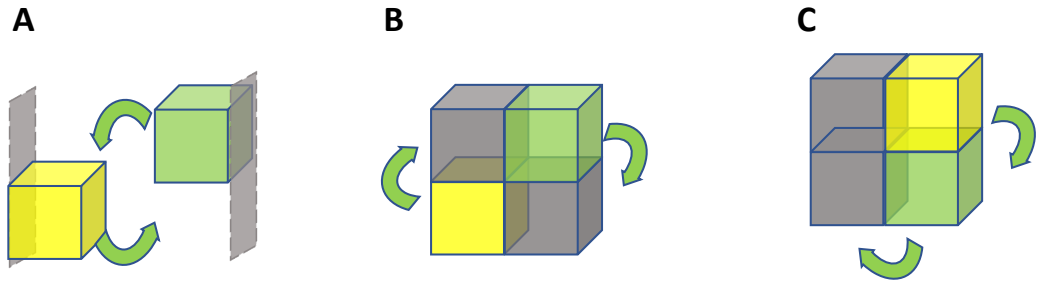


Figure 4. Illustration of two-agent conflict examples. Yellow and green modules are moving. The grays are static. (**A**) is an example of violating Specification 2. (**B**) is an example of violating Specification 3. (**C**) is an example of violating Specification 4.

3.3.2. Connectivity

Maintaining the overall connectivity of the configuration is a critical safety consideration during parallel reconfiguration. To meet this requirement, agents are prohibited from creating a cut in the connection graph of the current configuration. This ensures that, as modules undergo reconfiguration, the overall connectivity remains intact, preventing the potential disintegration of the MSRS. This is described as

Safety Specification 3. *(Connectivity) For two agents, v_i and v_j, their actions a^i and a^j follow* $\square(\neg((\mathcal{G} - v_i - v_j)$ *is connected*$) \rightarrow ((a^i = 0) \vee (a^j = 0)))$.

This means that two agents forming a cut in the configuration's connection graph cannot move simultaneously. Figure 4B is an example of violating the specification 3.

3.3.3. Stable Pivoting

There is a risk when one moving module designates another moving module as its supporting neighbor. This scenario introduces potential instability, as both modules are in motion simultaneously, violating the requirement that safe movement relies on a static supporting neighbor. To avoid this, safety specification is defined as

Safety Specification 4. *(Stable pivoting) For two agents, v_i and v_j, their supporting neighbors are v_i^n and v_j^n, respectively. Their actions a^i and a^j follow* $\square(((v_j = v_i^n) \vee (v_i = v_j^n)) \rightarrow ((a^i = 0) \vee (a^j = 0)))$.

Figure 4C is an example of violating the specification 4.

Remark 1. *A violation may occur if agent i attempts to select another agent j as its new neighbor after locomotion. If $\theta = 180°$, this scenario is already addressed by Specification 4. If $\theta = 90°$, another new neighbor position can either be empty, occupied by fixed module, or occupied by another agent k. The latter represents a typical conflict involving three agents: i, j, and k. For simplicity, this article does not formulate a specific safety specification for this case. Instead, we allow the agent to learn how to act through reward-based learning.*

4. Parallel Reconfiguration Planning Method

We propose a novel multi-agent reinforcement learning algorithm, termed the Parallel Reconfiguration Planner (PRP), designed to achieve a safe and efficient parallel reconfiguration policy (see Figure 5). As MMDP formulated (see Section 2.1), agents' actions jointly, rather than independently, determine rewards and state transitions in parallel reconfiguration. This interdependence necessitates a multi-agent reinforcement learning approach. PRP addresses this challenge by employing Value-Decomposed Q-learning

(VDN) as its backbone, where a neural network is trained for each agent to output its action in a decentralized manner (see Section 4.1). As input to the neural network, each agent has access to the full MSRS configuration as well as its own position within it. Parallel reconfiguration involves complex safety specifications, making conventional reinforcement learning approaches sample-inefficient due to the extensive exploration required to meet these specifications. To enhance exploration efficiency, PRP employs a centralized shielding mechanism for guarded exploration (see Section 4.2), selecting joint actions that satisfy the safety specifications outlined in (see Section 3) for execution. To further enhance sample efficiency, PRP integrates prioritized experience replay [45], enabling more effective sampling of training data (see Section 4.3).

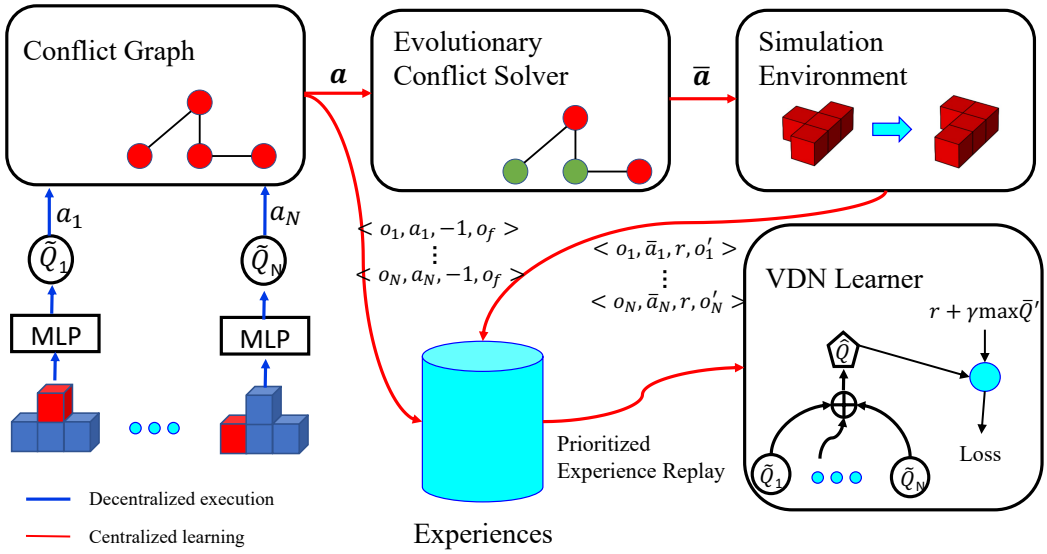


Figure 5. Illustration of parallel reconfiguration planning (PRP) algorithm. The blue line denotes the decentralized execution process, where each mobile module decides its action independently. The red line denotes the centralized training process, in which the Q-networks are trained in a centralized manner. The VDN learner, evolutionary conflict solver, and prioritized experience replay are introduced in Section 4.1, Section 2.2, and Section 4.3, respectively.

4.1. Multi-Agent Reinforcement Learning

As defined in the MMDP, each joint policy π has an associated action-value function $Q^{\pi}(c, \mathbf{a})$, where c represents the MSRS's configuration and $\mathbf{a}$ the joint action. The PRP aims to determine the parallel reconfiguration policy by finding the optimal joint policy π^*, whose action-value function corresponds to the optimal action-value function $Q^*(c, \mathbf{a})$. Derive from Equation (2):

$$Q^*(c, \mathbf{a}) = \max_{\pi} \mathbb{E}_{\pi, Pr}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid c_t = c, \mathbf{a}_t = \mathbf{a}\right] \tag{5}$$

$$= \mathbb{E}\left[r_{t+1} + \gamma \max_{\pi} \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid c_{t+1}\right] \mid c_t = c, \mathbf{a}_t = \mathbf{a}\right] \tag{6}$$

$$= \mathbb{E}[r_{t+1} + \gamma \max_{\mathbf{a}_{t+1}} Q^*(c_{t+1}, \mathbf{a}_{t+1}) | c_t = c, \mathbf{a}_t = \mathbf{a}] \tag{7}$$

Equation (7) is called the optimal Bellman equation of action-value. Q-learning [46] is a value iteration algorithm based on the Bellman equation, which iteratively updates the action-value function as

$$Q(c, \mathbf{a}) \leftarrow (1 - \beta) Q(c, \mathbf{a}) + \beta (r + \gamma \max_{\mathbf{a}'} Q(c', \mathbf{a}')). \tag{8}$$

where β is the learning rate. It has been proven that such value iterations converge to the optimal action-value function $Q^*(c, \mathbf{a})$. Consequently, the optimal reconfiguration policy is to select the action that maximizes the action-value function at every configuration: $\boldsymbol{\pi}^* = \underset{\mathbf{a}}{\text{argmax}} Q^*(c, \mathbf{a})$.

Implementing Q-learning in multi-agent scenarios presents two major limitations. First, Q-learning requires maintaining the value of each action in the joint action space, which grows exponentially with the number of agents. Second, each agent must know the actions of all other agents to make decisions, making distributed execution infeasible. To address these challenges, the PRP adopts Value-Decomposed Q-learning as its backbone. PRP decomposes the action-value function of the joint policy as

$$Q(c, \mathbf{a}) = \sum_{i=1}^{N} \tilde{Q}_i(o^i, a^i) \tag{9}$$

where $\tilde{Q}_i(o^i, a^i)$ is defined as the *independent* action-value function for agent i, a^i is action of agent i, and o^i is the observation of agent i. Instead of relying on a joint policy, the independent action-value function is determined by each agent's individual pivoting policy. Consequently, rather than centrally computing the action-value function for joint actions in the joint action space, PRP allows each agent to compute its independent action-value function for actions within its individual action space in a distributed manner, based solely on its local observation.

The decomposition in Equation (9) ensures that performing a global argmax on the action-value function produces the same result as applying argmax operations individually on each agent's independent action-value function:

$$\underset{\mathbf{a}}{\text{argmax}} Q(c, \mathbf{a}) = \begin{pmatrix} \underset{a^1}{\text{argmax}} \tilde{Q}_1(o^1, a^1) \\ \vdots \\ \underset{a^N}{\text{argmax}} \tilde{Q}_N(o^N, a^N) \end{pmatrix} \tag{10}$$

As a result, the optimal reconfiguration policy involves each agent selecting the action that maximizes its independent action-value function at every time step, based on the observation it receives.

To address the large state (observation) space in reconfiguration planning problems [47], PRP employs artificial neural networks (ANNs) to approximate independent action-value functions, referred to as Q-networks. Each agent is equipped with its own Q-network, $\hat{Q}_i(o^i, a^i|\hat{\omega}^i)$, where $\hat{\omega}^i$ represents the parameters of the Q-network. The joint action-value function is then approximated as $\hat{Q}(c, \mathbf{a}) = \sum_{i=1}^{N} \hat{Q}_i(o^i, a^i|\hat{\omega}^i)$. At each time step t, PRP collects the experience of each agent, $e_t^i = (o_t^i, a_t^i, r_{t+1}, o_{t+1}^i)$, through guarded exploration (see Section 4.2) and trains the Q-networks using a supervised learning approach.

Approximating the action-value function using ANNs is known to affect the stability of Q-learning algorithms [48]. Inspired by Deep Q-Learning (DQN, [49]), PRP incorporates *target networks* and *experience replay*, two signature techniques of DQN, to address such instability. To include target networks, each agent uses an additional neural network, $\bar{Q}_i(o^i, a^i|\bar{\omega}^i)$, referred to as the target network, with parameters $\bar{\omega}^i$. This yields the target action-value function: $\bar{Q}(c, \mathbf{a}) = \sum_{i=1}^{N} \bar{Q}_i(o^i, a^i|\bar{\omega}^i)$. To incorporate experience replay, PRP stores each agent's experience e_t^i at time step t in a dataset $D_t^i = \{e_1^i, \ldots, e_t^i\}$.

During learning, each agent sample minibatches of experiences $(o^i, a^i, r, o^{i\prime}) \sim D^i$ from their experience memory dataset using prioritized experience replay (see Section 4.3).

To perform the Q-learning update, the Q-networks are trained using backpropagation to minimize the loss function:

$$L(\hat{\omega}^1, \ldots, \hat{\omega}^N) = \mathbb{E}_{(o^i, a^i, r, o^{i\prime}) \sim D^i, i=1,\ldots,N}\left[loss(\hat{Q}(c, \mathbf{a}), y)\right]. \tag{11}$$

where $loss(\cdot)$ is Huber loss [50]:

$$loss(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| \leq \delta \\ \delta \cdot |y - \hat{y}| - \frac{\delta^2}{2}, & |y - \hat{y}| > \delta \end{cases}, \tag{12}$$

and y is the target of supervised learning, derived from Equation (8):

$$y = \begin{cases} r & \text{if } c' \text{ is a terminal state} \\ r + \gamma \max_{\mathbf{a}'} \bar{Q}(c', \mathbf{a}') & \text{otherwise} \end{cases}. \tag{13}$$

The Q-networks are updated at every time step once a sufficient number of experiences are accumulated in the memory. To ensure stability, the parameters of the target networks are periodically synchronized with the Q-network parameters after a fixed number of steps.

All agents use the same Q-network architecture. The Q-network takes the agent's observation as input, which consists of a matrix representation of the configuration and a position vector. These are concatenated and flattened into a single observation vector, which is then passed through a multi-layer perceptron (MLP) with one hidden layer and a leaky ReLU [51] activation function. The Q-network outputs the independent action-values for all actions in the agent's action space. To reduce the number of learnable parameters, PRP shares network weights between agents. This weight sharing introduces the concept of agent invariance, which aligns with module invariance in reconfiguration. With agent invariance, all agents exhibit identical policies under the same conditions, a property that is desirable in parallel reconfiguration.

4.2. Guarded Exploration

PRP uses guarded exploration (see line 10 to line 23 in Algorithm 1) to gather experiences for training. At the beginning of the guarded exploration, each agent independently computes its masked action space A_s^i using the Action Mask (AM, see Algorithm 2). The action space A_s^i contains only those actions that satisfy the single-agent safety specifications. The AM takes the observation o^i of each agent i as input and iterates through all actions in the action space to evaluate whether each action satisfies the *Safety Specification 1* . If an action meets the criteria, it is added to A_s^i.

After applying the AM, each agent i selects an action a^i from A_s^i using the ϵ-greedy policy. With probability ϵ, the agent chooses a random action, and with probability $1 - \epsilon$, it selects the action that maximizes its independent action-value function. Guarded exploration employs the ϵ-greedy policy as it effectively encourages exploration.

While these actions, forming a joint action $\mathbf{a} = (a^i, \ldots, a^N)$, can be executed safely on an individual basis, executing them simultaneously may violate safety specifications. Guarded exploration uses centralized shielding (CS, see Algorithm 3) to produce a revised joint action $\bar{\mathbf{a}}$, referred to as the shielded joint action, which satisfies the two-agent safety specifications (see Section 3.3). During guarded exploration, agents store not only transitions generated by executing shielded joint actions but also the original unsafe transitions. This leads to a distribution shift in the experience dataset. However, since VDN is an off-policy multi-agent reinforcement learning method, this distribution shift does not harm the training process.

Algorithm 1 Parallel reconfiguration planning.

Input: Initial configuration c_0 and goal configuration c_g
Output: Parallel reconfiguration policy $\boldsymbol{\pi}$

for agent $i = 1, N$ **do**
  Initialize replay memory D^i
  Initialize Q-network $\hat{Q}^i$ with random parameters $\hat{\omega}^i$
  Initialize target network $\bar{Q}^i$ with parameters $\bar{\omega}^i = \hat{\omega}^i$
end for
for episode = $1, K$ **do**
  Initialize configuration $c_1 = c_0$
  Set observation $o_1^i = (c_1, v_1^i)$ for all agent i
  for step $t = 1, T$ **do**
    for agent $i = 1, N$ **do**
      // *action masking*
      $A_s^i \leftarrow AM(o_t^i)$ (Algorithm 2)
      With probability ϵ sample an $a_t^i \sim U(A_s^i)$
      otherwise select $a_t^i = \underset{a_t^i \in A_s^i}{\operatorname{argmax}} \hat{Q}_i\left(o_t^i, a_t^i\right)$
    end for
    // *centralized shielding*
    $\bar{\mathbf{a}}_t \leftarrow CS(c_t, \mathbf{a}_t)$ (Algorithm 3)
    if $\bar{\mathbf{a}}_t \neq \mathbf{a}_t$ **then**
      Store (o_t^i, a_t^i, r_p, o_f) in D^i for all agent i
    end if
    Execute joint action $\bar{\mathbf{a}}_t$ in emulator and observe reward r_{t+1} and new configuration c_{t+1}
    Set observation $o_{t+1}^i = (c_{t+1}, v_{t+1}^i)$ for all agent i
    Store $(o_t^i, \bar{a}_t^i, r_{t+1}, o_{t+1}^i)$ in D^i for all agent i
    Sample minibatch of experiences $(o_j^i, a_j^i, r_{j+1}, o_{j+1}^i)$ from D^i using prioritize memory replay
    Set supervision target y_j with Equation (13)
    $\hat{Q}(c_j, \mathbf{a}_j) \leftarrow \sum_{i=1}^{N} \hat{Q}_j(o_j^i, a_j^i | \hat{\omega}^i)$
    Perform a gradient descent step on Huber loss between y_j and $\hat{Q}(c_j, \mathbf{a}_j)$ with respect to the network parameters $\hat{\omega}^1, \ldots, \hat{\omega}^N$
    Every C step reset $\bar{\omega}^i = \hat{\omega}^i$ for all agent i
    End current episode when observe a terminal state
  end for
end for
return $\boldsymbol{\pi} = \prod_i^N \underset{a^i}{\operatorname{argmax}} \hat{Q}_i\left(o^i, a^i\right)$

Algorithm 2 Action Mask (AM).

Input: Observation o^i
Output: Masked action space A_s^i

Initialize $A_s^i \leftarrow \emptyset$
for each action a in the action space A **do**
  if a satisfies Safety Specification 1 **then**
    Add a to A_s^i
  end if
end for
return A_s^i

The CS takes $\mathbf{a}$ and the configuration c as input. It first examines each pair of agents to determine whether their selected actions lead to conflicts (i.e., violations of safety Specifications 2, 3, or 4). Based on this analysis, it constructs a conflict graph $\mathcal{G}_c(\mathcal{V}_a, \mathcal{E}_c)$, where $\mathcal{V}_a$ represents the set of all agents, and $\mathcal{E}_c$ denotes the set of edges connecting agents in conflict (illustrated in Figure 6). CS produce $\bar{\mathbf{a}}$ by finding an optimal strategy where each agent selects a role as either a "hawk" or a "dove". A dove ceases its motion ($\hat{a}^i = 0$) to prevent conflict while a hawk proceeds with its pivoting motion ($\bar{a}^i = a^i$) to keep reconfiguration.

In this cooperative framework, a penalty is applied to all agents if two hawks are connected in the conflict graph $\mathcal{G}_c$, indicating unresolved conflicts. If no such conflicts exist (i.e., no connected hawks), the agents receive a payoff determined by the sum of their independent action-value functions: $\sum_{i=1}^{N} \hat{Q}_i(o^i, \bar{a}^i)$. Under this evaluation, the optimal hawk–dove strategy corresponds to the shielded joint action with the maximum action value, thereby aligning the exploration process with the Q-learning objective.

Algorithm 3 Centralized Shielding (CS).

Input: Configuration c, joint action $\mathbf{a}$
Output: Shielded joint action $\bar{\mathbf{a}}$
Check for conflicts between all pairs of agents
Build the conflict graph $\mathcal{G}_c$
if no conflicts exist in $\mathcal{G}_c$ **then**
    return $\bar{\mathbf{a}} = \mathbf{a}$
else
    // Use evolutionary conflict solver (ECS)
    $\bar{\mathbf{a}} \leftarrow ECS(\mathcal{G}_c)$ (see Algorithm 4)
    return $\bar{\mathbf{a}}$
end if

Inspired by Evolutionary Programming [52], CS employs an evolutionary conflict solver (see Algorithm 4) to determine the optimal strategy. Initially, all agents are assigned the hawk strategy. During each generation, the solver selects one agent to undergo mutation, switching its strategy either from hawk to dove or vice versa. The updated strategies are then evaluated based on their payoff. If the mutation results in a higher payoff, it is retained; otherwise, it may still be accepted with a probability determined by an annealing schedule. This iterative process continues for up to G generations, after which the final strategies are output (illustrated in Figure 6).

Algorithm 4 Evolutionary Conflict Solver.

Input: Conflict graph $\mathcal{G}_c$ of joint action $\mathbf{a}$
Output: Shielded joint action $\bar{\mathbf{a}}$
Initialize $\bar{\mathbf{a}} = \mathbf{a}$, heat $h = h_0$, and evaluation score E with a large penalty
for generation $g = 1$ to G **do**
    Randomly select one agent to mutate and generate a new joint action $\bar{\mathbf{a}}_g$
    Evaluate $\bar{\mathbf{a}}_g$:
    if two hawks are connected in $\mathcal{G}_c$ **then**
        Assign a large penalty to the evaluation
    else
        Compute payoff as $\sum_{i=1}^{N} \hat{Q}_i(o^i, \bar{a}_g^i)$
    end if
    if evaluation improves **or** with probability $\exp\left(\frac{\Delta E}{h}\right)$ **then**
        Accept mutation: $\bar{\mathbf{a}} = \bar{\mathbf{a}}_g$
    end if
    Update heat: $h \leftarrow \gamma_h h$
end for
return $\bar{\mathbf{a}}$



Figure 6. Illustration of the conflict graph and Evolutionary Conflict Solver (ECS). The red vertex represents the hawk agent, and the green vertex represents the dove agent. (**A**) illustrates the original strategy, where hawk agents are connected. (**B**) illustrates the strategy generated by the ECS, where hawk agents are only connected with dove agents.

4.3. Prioritized Experience Replay

PRP samples transition experiences $\mathbf{e}_j = (e_j^1, \ldots, e_j^N)$ from the experience dataset $\mathbf{D} = (D^1, \ldots, D^N)$, ensuring that all agents sample experiences corresponding to the same time step j. Instead of uniformly sampling from $\mathbf{D}$, PRP employs prioritized experience replay [45] to accelerate convergence. In prioritized experience replay, each experience is assigned a priority based on its *importance* to learning, with higher-priority experiences replayed more frequently. A practical proxy for importance is the TD error, defined as

$$\delta_j = r_{j+1} + \gamma \max_{\mathbf{a}_{j+1}} \hat{Q}(c_{j+1}, \mathbf{a}_{j+1}) - \hat{Q}(c_j, \mathbf{a}_j), \tag{14}$$

which measures how unexpected a transition is. PRP adopts rank-based prioritization, where transitions $\mathbf{e}_j$ are ranked by $|\delta_j|$, denoted as $\text{rank}(j)$. To mitigate sensitivity to noise, PRP uses stochastic sampling with probabilities defined as

$$P(j) = \frac{p_j^\alpha}{\sum p_k^\alpha}, \quad p_j = \frac{1}{\text{rank}(j)}, \tag{15}$$

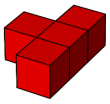
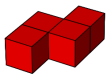
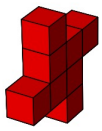
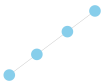
where α controls the degree of prioritization, resulting in a power-law distribution. This sampling balances greedy prioritization and random sampling to improve learning stability and performance.

5. Experiments

5.1. Scenarios and Evaluation Metrics

We evaluated our approach on three parallel reconfiguration tasks: (1) A **4-module MSRS with 2 agents (2A4M)**: The MSRS consists of four modules, two of which are movable and two are fixed. The initial configuration is a "Z" shape, and the goal configuration is a square shape. (2) A **4-module MSRS with 4 agents (4A4M)**: The MSRS consists of four movable modules. The initial configuration is a "T" shape, and the goal configuration is a square shape. (3) A **8-module MSRS with 8 agents (8A8M)**: The MSRS has eight modules, all of which are movable. The initial configuration is a "windmill" shape. The goal configuration is a cube shape. The illustrations of these three initial configurations are shown in Table 2. The "T" configuration can be applied to Earth observation missions, where the central module serves as the GNC module, and the remaining three modules function as two solar array modules and one camera module, respectively. The "windmill" configuration is suitable for on-orbit takeover tasks: the four central modules provide the docking structure to interface with the target spacecraft, while the four surrounding modules act as thrust modules to generate attitude control torque. The square and cube configurations are applicable to on-orbit repair missions; in these configurations, each module can be safely removed, which facilitates MSRS maintenance by replacing faulty modules.

Table 2. Examples of configuration representation.

Configuration	"T" Shape	"Z" Shape	"Windmill" Shape
Illustration			
Matrix representation	$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & 3 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 2 & 1 \end{bmatrix}$
Graph representation			

We propose two evaluation metrics. The first metric, **Normalized Score (Norm.Score)**, is based on the total rewards collected during an episode, referred to as the episode's *Score*. For each scenario, the episode score lies within the range $[Score_{min}, r_g)$. We use the score of the worst episode encountered during training as the $Score_{min}$. **Norm.Score** is defined as $\frac{Score - Score_{min}}{r_g - Score_{min}}$. Before training begins, we collect a fixed set of agents' observations by running a random policy from the initial configuration. From this set, we randomly sample a subset called the held-out set of observations. The second metric, **Average Action Value (Average.Q)**, is defined as the average of the maximum predicted independent action values for the observations in the held-out set.

5.2. Our PRP Method Enables Parallel Reconfiguration

Although MSRSs are designed to move multiple modules simultaneously, existing reconfiguration planning algorithms can only make sequential, one-by-one motion decisions. We evaluated our PRP on three 3D reconfiguration scenarios. The Adam optimizer [53], an adaptive stochastic optimization algorithm that computes individual learning rates for each parameter based on estimates of the first and second moments of the gradients, is employed to update the parameters of the Q-network. By combining the advantages of momentum and RMSProp, Adam provides efficient and stable convergence in large-scale and noisy optimization problems. Table 3 lists the hyperparameters for training. The performance of the MSRS using PRP across different scenarios is shown Figures 7 and 8. As training progresses, both the Norm.Score and the Average.Q increase steadily, indicating that the training is progressing successfully. The generated reconfiguration policies are illustrated in Figure 9. The results shows that PRP can make simultaneous pivoting decisions, truly enabling MSRS modules to move in parallel. Moreover, the three evaluation scenarios correspond to different initial configurations of the MSRS. For each scenario, a separate model is trained with the corresponding initial configuration and evaluated on the same initial configuration during testing. The results demonstrate that the proposed learning algorithm can effectively solve reconfiguration tasks under different predefined starting states.

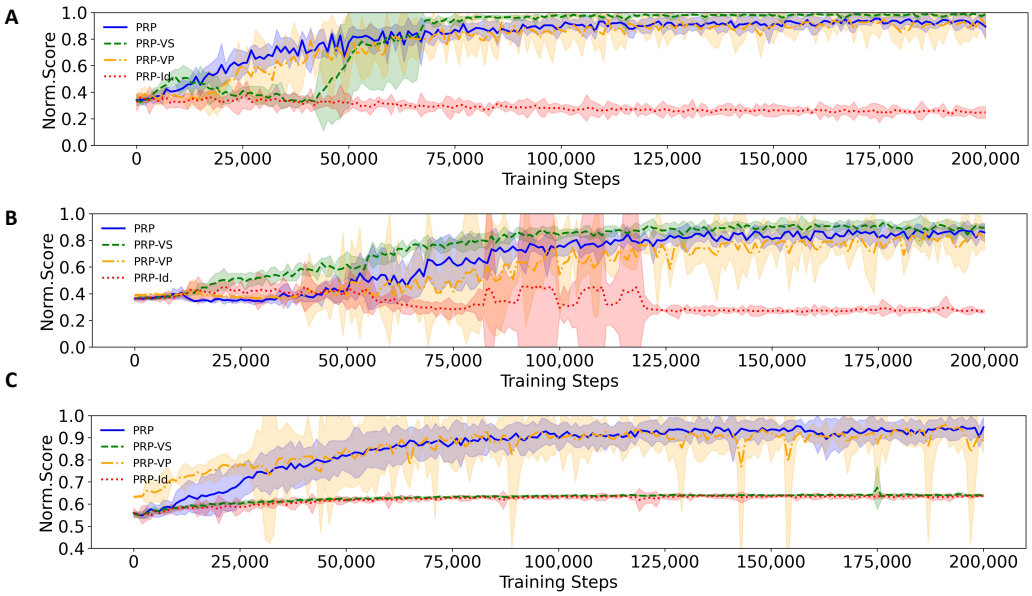


Figure 7. Performance of MSRS using PRP and each ablated algorithms in each scenarios. We report how normalized score changes during learning process (random seeds = 0, 42, 123, 2147483647, and 3491726381). The shaded areas indicate the 95% confidence intervals. (**A**–**C**) are the Norm.Score of 2A4M, 4A4M, and 8A8M, respectively. See Section 5.1 for Norm.Score and reconfiguration scenarios, and Section 5.4 for the ablated algorithms.



Figure 8. Average.Q of MSRS using PRP and each ablated algorithms in each scenarios. We report how Average Action Value (Average.Q) changes during learning process (random seeds = 0, 42, 123, 2147483647, and 3491726381). The shaded areas indicate the 95% confidence intervals. (**A**–**C**) are the Average.Q of 2A4M, 4A4M, and 8A8M, respectively. See Section 5.1 for Average.Q and reconfiguration scenarios, and Section 5.4 for the ablated algorithms.

Table 3. Hyperparameters for PRP training.

Hyperparameter	Value
Goal reward r_g	10
Penalty reward r_p	-1
Reward constant r_i	15
Learning rate	1×10^{-4}
Replay-buffer size	800
Discount factor γ	0.99
Batch size	32
Max mutation generation G	100
Initial heat h_0	100
Annual rate γ_h	0.9

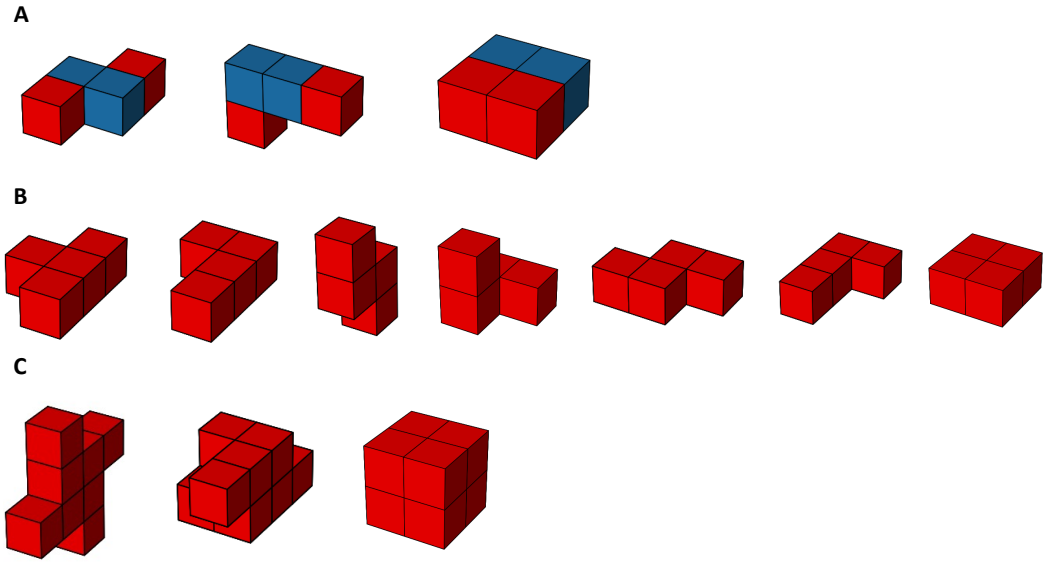


Figure 9. Illustration of the reconfiguration policy generated by PRP.The red cubes are movable modules. The blue cubes are fixed modules. (**A**–**C**) are the reconfiguration policies of 2A4M, 4A4M, and 8A8M, respectively.

5.3. Our PRP Method Outperforms Baseline Methods

We include three baseline methods for comparison. (1) ***RECONFIGURE3D***: a centralized, rule-based reconfiguration planning method proposed in [12]. This algorithm employs a line configuration as an intermediate step. *RECONFIGURE3D* plans reversible paths for both the initial and goal configurations to reconfigure to line configuration. It first reconfigures the system into a line configuration and then follows the reverse path to transition from the line configuration to the goal configuration. (2) **A***: a search-based planning algorithm. We follow the implementation proposed in [5] to adapt it to the reconfiguration planning task. (3) **Proximal Policy Optimization (PPO)**: a single-agent deep reinforcement learning method proposed in [54]. When adapting PPO to reconfiguration planning, the MSRS is treated as a centralized agent that moves only one module at a time. Table 4 presents the normalized score (Norm.Score) of PRP compared to baseline methods. PRP outperforms *RECONFIGURE3D* across all three scenarios. Both A* and PPO can only generate sequential reconfiguration plans, in which modules move one at a time. They produce identical trajectories across all three scenarios, corresponding to the optimal one-by-one reconfiguration paths. As centralized planning methods, A* and PPO outperform PRP in the 2A4M and 4A4M scenarios, where PRP operates in a decentralized manner and therefore faces greater difficulty in learning optimal solutions. In contrast,

PRP outperforms A* and PPO in the 8A8M scenario, as PRP explicitly supports parallel reconfiguration, enabling more efficient.

Table 4. Comparison of Norm.Score with baseline methods. The bold numbers denote the highest values for each scenario.

	2A4M	**4A4M**	**8A8M**
RECONFIGURE3D	0.78	0.80	−1.65
A*	**0.99**	**0.99**	0.97
PPO	**0.99**	**0.99**	0.97
Ours (PRP)	0.98	0.96	**0.99**

The optimal assignment metric, introduced by [43], is employed during the reconfiguration process to assess the efficiency of different methods (see Figure 10). A value of zero indicates that the reconfiguration is complete. The experimental results show that *RECONFIGURE3D* produces the longest reconfiguration trajectories across all three scenarios. Its optimal assignment metric initially increases and then decreases, as the method first reconfigures the system into a line configuration, which reduces the overall efficiency of the reconfiguration process. In the 2A4M setting, PRP, A*, and PPO complete reconfiguration in the same number of steps, as all three methods yield optimal policies. In the 4A4M setting, PRP requires more reconfiguration steps than A* and PPO, since its decentralized nature makes learning optimal reconfiguration policies more challenging, whereas A* and PPO, as centralized planning methods, can generate optimal solutions. In contrast, in the 8A8M setting, the model trained with the proposed PRP achieves the shortest reconfiguration trajectories by enabling parallel reconfiguration.

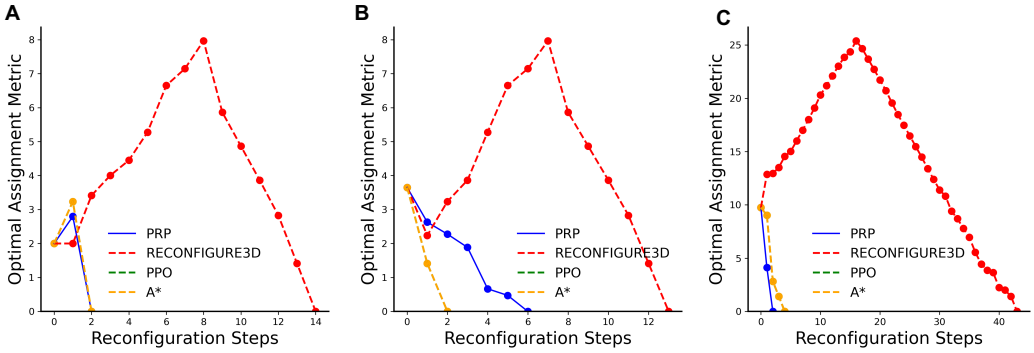


Figure 10. The optimal assignment metric between the goal configuration and the MSRS configuration at each reconfiguration step. Solid blue lines denote the test-time performance of PRP. The red, green, and yellow dashed lines represent the performance of reconfiguration policies generated by *RECONFIGURE3D*, A*, and PPO, respectively. (**A**–**C**) correspond to the performances of 2A4M, 4A4M, and 8A8M, respectively.

5.4. Ablation Study Reveals Critical Component Designs

We systematically ablated several key components of our parallel reconfiguration algorithm and present the results in Table 5 and Figure 7. **PRP** refers to the full algorithm. **PRP-Id.** is an algorithm where agents are trained independently using DQN, sharing network parameters and experiences, and utilizing both guarded exploration and prioritized experience replay. This ablated algorithm failed in all three scenarios, indicating that value decomposition is crucial for the stable convergence of learning. **PRP-VS** is the full algorithm with prioritized experience replay removed. The failure in the 8A8M scenario indicates that prioritized experience replay is essential for scenarios with a larger number of agents. **PRP-VP** is the full algorithm with centralized shielding removed. The failure in

the 8A8M scenario indicates that the guarded exploration mechanism is effective for better parallel reconfiguration planning.

Table 5. Comparison of Norm.Score with ablated methods. The bold numbers denote the highest values for each scenario. The symbol "-" indicates that the model failed to reconfigure to the target configuration during testing.

	2A4M	**4A4M**	**8A8M**
PRP-VS	**0.99**	0.98	-
PRP-VP	0.98	0.96	-
PRP-Id.	-	-	-
Ours (PRP)	0.98	0.96	**0.99**

6. Conclusions

This article introduces the Parallel Reconfiguration Planner (PRP), a reconfiguration planning algorithm for modular self-reconfigurable satellites (MSRSs) that enables multiple modules to move simultaneously, thus improving reconfiguration efficiency. The article uses Linear Temporal Logic (LTL) to define the safety specifications during parallel reconfiguration. Evaluation results on 3D reconfiguration scenarios demonstrate that in the larger module number setting, PRP outperforms the baseline method, which requires modules to move sequentially. PRP comprises three key components: a value-decomposed Q-learning as the backbone of the multi-agent reinforcement learning algorithm, guarded exploration, and prioritized experience replay. The ablation study confirms that each component contributes to the success of parallel reconfiguration planning. While the proposed PRP demonstrates the effectiveness of multi-agent reinforcement learning for parallel reconfiguration planning, several limitations remain. First, the current formulation assumes perfect state observability; incorporating partial observability would improve robustness and better reflect real-world deployment conditions, such as the presence of sensor noise or communication delays. In future work, methods using fuzzy logic and Hidden Markov Models [55,56] could be incorporated to address partially observable problems. Second, the proposed method is evaluated under fixed starting and goal configurations. Developing a model capable of generalizing to a broader range of unseen starting and goal configurations represents an important research direction for future work. Third, because the cooperative behavior of agents becomes harder to learn as the number of moving modules gets higher, training PRP for larger systems incurs substantial computational cost. Future work will focus on developing more efficient learning algorithms and studying the scalability of the learning algorithms.

Author Contributions: Conceptualization, B.W. and D.Y.; methodology, B.W.; software, B.W., S.L. and J.Y.; validation, B.W., S.L. and J.Y.; formal analysis, B.W.; investigation, B.W.; resources, D.Y. and Z.S.; data curation, B.W., S.L. and J.Y.; writing—original draft preparation, B.W.; writing—review and editing, D.Y. and Z.S.; visualization, B.W.; supervision, D.Y. and Z.S.; project administration, D.Y. and Z.S.; funding acquisition, D.Y. and Z.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

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

References

1. Seo, J.; Paik, J.; Yim, M. Modular reconfigurable robotics. *Annu. Rev. Control. Robot. Auton. Syst.* **2019**, *2*, 63–88. [CrossRef]
2. Oberländer, J.; Uhl, K.; Pfotzer, L.; Göller, M.; Rönnau, A.; Dillmann, R. Management and manipulation of modular and reconfigurable satellites. In Proceedings of the ROBOTIK 2012, 7th German Conference on Robotics, VDE, Munich, Germany, 21–22 May 2012; pp. 1–6.
3. Goeller, M.; Oberlaender, J.; Uhl, K.; Roennau, A.; Dillmann, R. Modular robots for on-orbit satellite servicing. In *Proceedings of the 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), Guangzhou, China, 11–14 December 2012*; IEEE: New York, NY, USA, 2012; pp. 2018–2023.
4. Melroy, P.; Hill, L.; Fowler, E.E.; Hunter, R.; Eagen, J.; Sullivan, B.R.; Will, P.; Palmer, J. DARPA phoenix satlets: Progress towards satellite cellularization. In Proceedings of the AIAA SPACE 2015 Conference and Exposition, Pasadena, CA, USA, 31 August–2 September 2015; p. 4487.
5. Song, Q.; Ye, D.; Sun, Z.; Wang, B. Motion planning techniques for self-configuration of homogeneous pivoting cube modular satellites. *Aerosp. Sci. Technol.* **2022**, *120*, 107249. [CrossRef]
6. Ye, D.; Wang, B.; Wu, L.; del Rio Chanona, E.A.; Sun, Z. PO-SRPP: A Decentralized Pivoting Path Planning Method for Self-Reconfigurable Satellites. *IEEE Trans. Ind. Electron.* **2024**, *71*, 14318–14327. [CrossRef]
7. Breon, S.; Boyle, R.; Francom, M.; DeLee, C.; Francis, J.; Mustafi, S.; Barfknecht, P.; McGuire, J.; Krenn, A.; Zimmerli, G.; et al. Robotic refueling mission-3—An overview. In *IOP Conference Series: Materials Science and Engineering*; IOP Publishing: Bristol, UK, 2020; Volume 755, p. 012002.
8. Letier, P.; Yan, X.T.; Deremetz, M.; Bianco, A.; Grunwald, G.; Roa, M.; Krenn, R.; Arancón, M.M.; Dissaux, P.; Casarrubios, J.S.G.; et al. MOSAR: Modular spacecraft assembly and reconfiguration demonstrator. In Proceedings of the 15th Symposium on Advanced Space Technologies in Robotics and Automation, Noordwijk, The Netherlands, 27–28 May 2019.
9. Battista, T.A.; Helvajian, H. A simulation architecture for self-assembling and reconfigurable space structures. In Proceedings of the AIAA Scitech 2021 Forum, Virtual, 11–15 and 19–21 January 2021; p. 1793.
10. Helvajian, H. Doing much with little: Hive a new space architecture. In Proceedings of the ASCEND 2020, Virtual, 16–18 November 2020; p. 4172.
11. Li, D.; Zhong, L.; Zhu, W.; Xu, Z.; Tang, Q.; Zhan, W. A survey of space robotic technologies for on-orbit assembly. *Space Sci. Technol.* **2022**, *2022*, 9849170. [CrossRef]
12. Sung, C.; Bern, J.; Romanishin, J.; Rus, D. Reconfiguration planning for pivoting cube modular robots. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 1933–1940.
13. Romanishin, J.W.; Gilpin, K.; Rus, D. M-blocks: Momentum-driven, magnetic modular robots. In Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, 3–7 November 2013; pp. 4288–4295.
14. Romanishin, J.W.; Gilpin, K.; Claici, S.; Rus, D. 3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 1925–1932.
15. Heidt, H.; Puig-Suari, J.; Moore, A.; Nakasuka, S.; Twiggs, R. CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation. In Proceedings of the 14th Annual AIAA/USU Small Satellite Conference, Logan, UT, USA, 21–24 August 2000.
16. Spröwitz, A.; Moeckel, R.; Vespignani, M.; Bonardi, S.; Ijspeert, A.J. Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot. *Robot. Auton. Syst.* **2014**, *62*, 1016–1033. [CrossRef]
17. Liedke, J.; Matthias, R.; Winkler, L.; Wörn, H. The collective self-reconfigurable modular organism (CoSMO). In Proceedings of the 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Wollongong, NSW, Australia, 9–12 July 2013; pp. 1–6.
18. Paredis, C.J.; Brown, H.B.; Khosla, P.K. A rapidly deployable manipulator system. *Robot. Auton. Syst.* **1997**, *21*, 289–304. [CrossRef]
19. Farritor, S.; Dubowsky, S. On modular design of field robotic systems. *Auton. Robot.* **2001**, *10*, 57–65. [CrossRef]
20. Jing, G.; Tosun, T.; Yim, M.; Kress-Gazit, H. An End-To-End System for Accomplishing Tasks with Modular Robots. In Proceedings of the Robotics: Science and Systems, Ann Arbor, MI, USA, 18–22 June 2016; Volume 2.
21. Feshbach, D.; Sung, C. Reconfiguring non-convex holes in pivoting modular cube robots. *IEEE Robot. Autom. Lett.* **2021**, *6*, 6701–6708. [CrossRef]
22. Zhang, C.; Zhou, L.; Li, Y. Pareto optimal reconfiguration planning and distributed parallel motion control of mobile modular robots. *IEEE Trans. Ind. Electron.* **2024**, *71*, 9255–9264. [CrossRef]
23. Wang, X.; Zhang, B.; Shang, W.; Cong, S. Optimal reconfiguration planning of a 3-DoF point-mass cable-driven parallel robot. *IEEE Trans. Ind. Electron.* **2023**, *71*, 2672–2682. [CrossRef]
24. Xiao, N.; Xiao, Y.; Ye, D.; Sun, Z. Adaptive differential game for modular reconfigurable satellites based on neural network observer. *Aerosp. Sci. Technol.* **2022**, *128*, 107759. [CrossRef]

25. Liu, W.; Geng, Y.; Wu, B.; Biggs, J.D. Distributed constrained control allocation for cellularized spacecraft attitude control system. *J. Guid. Control Dyn.* **2022**, *45*, 385–393. [CrossRef]
26. Shi, K.; Sun, Z.; Liu, C.; Ye, D. Dynamics modeling and simulation of space electromagnetic docking for CubeSat. In Proceedings of the 2017 8th International Conference on Mechanical and Aerospace Engineering (ICMAE), Prague, Czech Republic, 22–25 July 2017; pp. 603–608.
27. Liu, B.; Ye, D.; Yao, Z.; Sun, Z. A Goal-Oriented Reinforcement Learning-Based Path Planning Algorithm for Modular Self-Reconfigurable Satellites. *IFAC-PapersOnLine* **2025**, *59*, 1446–1451. [CrossRef]
28. Chen, L.; Huo, M.; Zhao, X.; Shang, X.; Yao, L.; Yang, D.; Fan, Z.; Qi, N. Distributed Reconfiguration Planning Optimization Method of Large-Scale Structure for Modular Satellites. *Adv. Astronaut.* **2025**, *8*, 351–359. [CrossRef]
29. Chen, L.; Qi, N.; Cheng, X.; Huo, M.; Yuan, Q.; Sun, K.; Yao, B.; Wang, L. Large-Scale Rapid Growth Reconfiguration in Space: A Decentralized Self-Reconfiguration Motion Planning Optimization Strategy for Space Modular Self-Reconfigurable Spherical Satellites. *IEEE Trans. Aerosp. Electron. Syst.* **2025**, *61*, 9359–9373. [CrossRef]
30. Chen, L.; Qi, N.; Huo, M.; Yuan, Q.; Yu, Z.; Feng, W. Growing fast and steady in space: Distributed rapid self-reconfiguration motion planning optimization methods for swarm intelligent space modular self-reconfigurable satellites. *Aerosp. Sci. Technol.* **2025**, *166*, 110536. [CrossRef]
31. Yu, J.; Ye, D.; Liu, H.; Guo, P.; Meng, L. Artificial potential field-based cluster motion planning method for modular self-reconfigurable satellites. *Aerosp. Sci. Technol.* **2026**, *176*, 112134. [CrossRef]
32. He, T.; Zhang, C.; Xiao, W.; He, G.; Liu, C.; Shi, G. Agile but safe: Learning collision-free high-speed legged locomotion. *arXiv* **2024**, arXiv:2401.17583.
33. Hansen, N.; SV, J.; Sobal, V.; LeCun, Y.; Wang, X.; Su, H. Hierarchical World Models as Visual Whole-Body Humanoid Controllers. *arXiv* **2024**, arXiv:2405.18418.
34. Andrychowicz, O.M.; Baker, B.; Chociej, M.; Jozefowicz, R.; McGrew, B.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; Ray, A.; et al. Learning dexterous in-hand manipulation. *Int. J. Robot. Res.* **2020**, *39*, 3–20. [CrossRef]
35. Kaufmann, E.; Bauersfeld, L.; Loquercio, A.; Müller, M.; Koltun, V.; Scaramuzza, D. Champion-level drone racing using deep reinforcement learning. *Nature* **2023**, *620*, 982–987. [CrossRef] [PubMed]
36. Song, Q.; Ye, D.; Sun, Z.; Wang, B. Autonomous reconfiguration of homogeneous pivoting cube modular satellite by deep reinforcement learning. *Proc. Inst. Mech. Eng. Part I J. Syst. Control Eng.* **2021**, *235*, 1777–1786. [CrossRef]
37. Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W.M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J.Z.; Tuyls, K.; et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv* **2017**, arXiv:1706.05296.
38. Lowe, R.; Wu, Y.I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*; NIPS: San Diego, CA, USA, 2017; Volume 30.
39. Rashid, T.; Samvelyan, M.; De Witt, C.S.; Farquhar, G.; Foerster, J.; Whiteson, S. Monotonic value function factorisation for deep multi-agent reinforcement learning. *J. Mach. Learn. Res.* **2020**, *21*, 1–51.
40. Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; Whiteson, S. Counterfactual multi-agent policy gradients. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
41. Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A.; Wu, Y. The surprising effectiveness of ppo in cooperative multi-agent games. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24611–24624. [CrossRef]
42. ElSayed-Aly, I.; Bharadwaj, S.; Amato, C.; Ehlers, R.; Topcu, U.; Feng, L. Safe multi-agent reinforcement learning via shielding. *arXiv* **2021**, arXiv:2101.11196.
43. Pamecha, A.; Ebert-Uphoff, I.; Chirikjian, G. Useful metrics for modular robot motion planning. *IEEE Trans. Robot. Autom.* **1997**, *13*, 531–545. [CrossRef]
44. Pnueli, A. The temporal logic of programs. In Proceedings of the 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), Providence, RI, USA, 31 October–2 November 1977; pp. 46–57.
45. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.
46. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
47. Chen, I.M.; Burdick, J.W. Enumerating the non-isomorphic assembly configurations of modular robotic systems. *Int. J. Robot. Res.* **1998**, *17*, 702–719. [CrossRef]
48. Tsitsiklis, J.; Van Roy, B. Analysis of temporal-diffference learning with function approximation. *Adv. Neural Inf. Process. Syst.* **1996**, *9*, 1075–1081.
49. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
50. Huber, P.J. Robust estimation of a location parameter. In *Breakthroughs in Statistics: Methodology and Distribution*; Springer: Berlin/Heidelberg, Germany, 1992; pp. 492–518.
51. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier nonlinearities improve neural network acoustic models. In Proceedings of the 30th International Conference on Machine Learning, Atlanta, GA, USA, 17–19 June 2013; Volume 30, p. 3.

52. Fogel, D.B.; Fogel, L.J. An introduction to evolutionary programming. In *European Conference on Artificial Evolution*; Springer: Berlin/Heidelberg, Germany, 1995; pp. 21–33.
53. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015.
54. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
55. Ji, W.; Ma, M.; Qiu, J.; Lam, H.K. Asynchronous fuzzy dynamic sliding mode control for nonlinear Markov jump systems under hidden mode detections. *IEEE Trans. Syst. Man Cybern. Syst.* **2025**, *55*, 7694–7704. [CrossRef]
56. Ji, W.; Chen, H.; Qiu, J.; Fan, Y. Output feedback asynchronous fuzzy SMC of nonlinear Markov jump systems via hidden mode detections. *IEEE Trans. Syst. Man Cybern. Syst.* **2026**, *56*, 3049–3059. [CrossRef]

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