TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 14/24 pp1481–1494
DOI: 10.26599/TST.2023.9010117
Volume 29, Number 5, October 2024

# Reset-Free Reinforcement Learning via Multi-State Recovery and Failure Prevention for Autonomous Robots

Xu Zhou, Benlian Xu*, Zhengqiang Jiang, Jun Li, and Brett Nener

**Abstract:** Reinforcement learning holds promise in enabling robotic tasks as it can learn optimal policies via trial and error. However, the practical deployment of reinforcement learning usually requires human intervention to provide episodic resets when a failure occurs. Since manual resets are generally unavailable in autonomous robots, we propose a reset-free reinforcement learning algorithm based on multi-state recovery and failure prevention to avoid failure-induced resets. The multi-state recovery provides robots with the capability of recovering from failures by self-correcting its behavior in the problematic state and, more importantly, deciding which previous state is the best to return to for efficient re-learning. The failure prevention reduces potential failures by predicting and excluding possible unsafe actions in specific states. Both simulations and real-world experiments are used to validate our algorithm with the results showing a significant reduction in the number of resets and failures during the learning.

**Key words:** reinforcement learning; manual reset; multi-state recovery; failure prediction

## 1 Introduction

Autonomous robots are expected to safely operate in unknown environments for extended lengths of time without human intervention. Reinforcement learning (RL) holds promise for robot autonomy because it can adapt to various environments by trial and error learning from the interactions between robots and surrounding environments. However, many RL algorithms require manually resetting the state of the system between training episodes[1–3]. For example, robots using episodic RL need human intervention to be reset to the same initial state for a new learning episode if they experience failures during the learning. Since manual resets are not desired in robot autonomy, it is practically important to design an autonomous reset method for RL.

Manual resets are needed in many situations such as experiencing failures or successfully finishing an episode and transiting from the goal state of one episode to the initial state of the next episode. Because the failure is a main and common reason of the reset, we mainly focus on the failure-induced resets in this work. It is then intuitive to design the autonomous reset method from two perspectives: (1) Provide reasonable recovery plans for failures that have already occurred. Instead of simply abandoning the failure episode that results in resetting to the initial state, the recovery plans can oppositely continue the learning to make the failure episode complete and successful in achieving

• Xu Zhou is with School of Mechanical Engineering, Changshu Institute of Technology, Changshu 215500, China. E-mail: xuzhou@cslg.edu.cn.
• Xu Zhou and Jun Li are with School of Automation, Nanjing University of Science and Technology, Nanjing 210094, China. E-mail: lijun1008@163.com.
• Benlian Xu is with School of Electronic and Information Engineering, Suzhou University of Science and Technology, Suzhou 215009, China. E-mail: xu_benlian@usts.edu.cn.
• Zhengqiang Jiang is with Faculty of Medicine and Health, The University of Sydney, Sydney 2006, Australia. E-mail: zhengqiang.jiang@sydney.edu.au.
• Brett Nener is with Department of Electrical, Electronic and Computer Engineering, The University of Western Australia, Perth 6009, Australia. E-mail: brett.nener@uwa.edu.au.
* To whom correspondence should be addressed.
Manuscript received: 2023-08-13; revised: 2023-10-02; accepted: 2023-10-10

the training goal. This not only substantially removes the necessity of resets, but may also assist expediting the learning process. (2) Prevent potential failures from happening in the learning process. Complementary to the recovery plans, this reduces the necessity of resets in terms of the frequency. In other words, fewer failures mean fewer resets.

Consider an example scenario in which an autonomous robot uses RL to navigate in an unknown maze. At a specific moment, the robot may face an obstacle in front of it and RL tells the robot to go forward. This situation seemingly presents a quandary: RL is waiting for the robot to reach the state of the obstacle to continue the remaining steps in current episode, but the robot cannot because it keeps hitting the obstacle and may be actually waiting for RL to solve this problem such as giving another feasible instruction.

RL can use external failure recovery methods[4−7] to diagnose the failure and bypass it with alternative plans. In other words, when a failure happens, RL stops working and the external failure recovery method takes over the robot control and recovers the robot back to a safe state or the state that RL wants to go. When the robot is safe, RL resumes working independently. However, such an external failure recovery method can only correct the mistake whenever it happens but cannot analyze why it happens and hence RL can continue making the same mistake. It is then ideal for RL itself to have the capability of generating recovery plans to get rid of the problematic state.

More importantly, the recovery plan should be able to determine which state to go back to. It could be neither necessary nor the best to recover back to the initial states as most RL algorithms do. For instance, if a robot reaches a dead end in a maze, it may be better to recover back to the junction to other paths rather than the starting position considering that the junction may be closer to the goal position and the remaining exploration is more efficient with taking less time. On the other hand, it is obviously meaningless and unreasonable to recover back to the position that is still in the dead end.

Manual resets could be also avoided by predicting failures that result in resets and preventing them from happening. For RL, this means predicting unsafe actions in advance and excluding them during the exploration. The prediction can be achieved by using prior knowledge for a known environment. However, when the environment is unknown or dynamic, the prediction should be based on the information or knowledge learned online by the robot itself.

To this end, we propose a reset-free reinforcement learning (RFRL) algorithm that can recover from failures to multiple previous safe states and prevent potential failures. More specifically, the multi-state recovery (MSR) evaluates the safety of previous states and then determines which state is the best for recovery. The failure prevention (FP) predicts unsafe actions that can result in failures and excludes them during the exploration. The contributions are three-fold. Firstly, a new reinforcement learning algorithm based on multi-state recovery and failure prevention is proposed to solve the manual reset problem in practical deployment of RL on autonomous robots. The MSR endows RL an important capability of dealing with the failures that have already happened, which is of great importance in practice but has not been investigated yet. Secondly, the proposed failure prevention can be regarded as a practical safe exploration method that can work flexibly with or without prior knowledge. For an unknown environment, the proposed FP uses a simple method of recording and prohibiting unsafe actions that cause failures. If the environment is known or partially known, prior knowledge can be used in generating a failure prediction model to provide a more accurate prediction of unsafe actions. Lastly, both simulation results and real-world experiment results are provided to demonstrate that either multi-state recovery or failure prevention can individually reduce resets while their combination, RFRL, can further reduce resets in a more efficient way.

## 2 Related Work

RL has achieved success in robot autonomy[8−12] in recent years. In general, RL methods can be divided into two categories: model-based methods[13] and model-free methods[14]. Model-based methods are known to be sample-efficient and model-free methods are popular due to their simplicity and favorable computational properties. However, the practical deployment of both methods often assumes an episodic setting, which requires manually resetting the state of the system between episodes[2]. Since manual resets are not desired in robot autonomy, our work aims to solve this important problem by proposing an autonomous reset method, including self-recovering from failures and reducing potential failures.

Although current safe RL algorithms[15] do not specifically focus on solving the reset problem, they can indeed reduce the resets by reducing the failures in RL. Some methods add safety concerns into the original RL optimization criteria[16–19] to rule out unsafe policies, and other methods restrict explorations[20–23] by adding safety constraints on RL action selections. Robust RL algorithms[24–26] can also improve the safety of RL due to their robustness to environment dynamics by identifying uncertain components of the environment and randomizing the parameters based on domain randomization. While these methods can indirectly reduce the resets, the safety concerns or constraints in them still require designs by humans, which may not be always available in practical applications. In addition, these methods do not investigate the recovery from failures that have occurred. When failures cannot be avoided, these methods need manual resetting again.

While a few recent works[1–3] specifically target on learning an automatic reset policy for RL, they also lack the capability of dealing with failures that cannot be avoided to happen. In addition, the reset policies learned in previous works[1–3] generally reset to the initial state, which may not always be necessary or beneficial. Our work investigates which previous safe state can be the best return state.

## 3 Methodology

As shown in Fig. 1, our reset-free reinforcement learning consists of two key parts, multi-state recovery and failure prevention. The multi-state recovery evaluates the safety level of previous states and determines which state is the best to reset to when a failure is detected. Instead of simply abandoning the failed episode with resetting to the initial state, the multi-state recovery aims to finish the training goal in each episode. The failure prevention uses failure recording for an unknown environment or a few-shot learning model for a known or partially known environment to predict and exclude unsafe actions, which prevents failures from occurring during the learning process. For simplicity, we use a standard $\varepsilon$-greedy $Q$-learning algorithm[27] as our basic RL algorithm for illustration. However, our method is actually designed for general RL methods because multi-state recovery and failure prevention are both based on general definitions of states and actions in RL.

### 3.1 Multi-state recovery

When a robot experiences a failure in a problematic state in practice, it is possible to recover back to a previous, unproblematic state and make a new plan. To make this recovery practically applicable, we assume that there exists an available motion controller for the robot to execute the recovery plan such as going to a certain state.

An important question in the recovery is: Which state is the best to recover back to? Intuitively, such a state

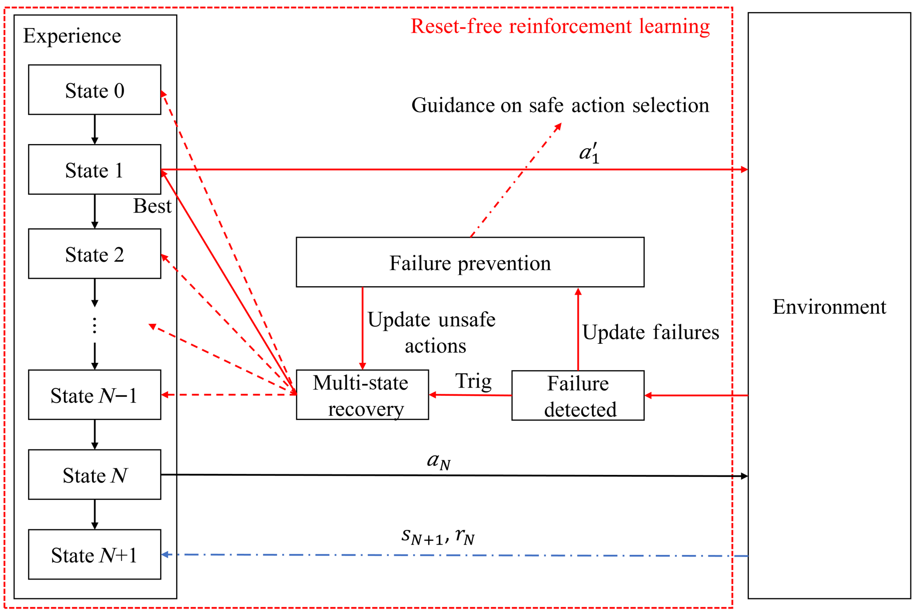

**Fig. 1 Schematic diagram of reset-free reinforcement learning.**

should be as safe as possible and can assist the subsequent learning as efficiently as possible. During a specific episode, for example, the robot is in the grey state and taking a problematic action "go south" as shown in Fig. 2. Because the state that the robot wants to enter is an obstacle, a failure will occur and be detected. It may then be ideal for the robot to explore other actions in the current position if it does not know the results of these alternative actions. Not doing this and directly going back to other states such as the previous light blue state may sometimes miss an optimal path if the block right to the grey state is not an obstacle. However, if the robot has already known the consequences of alternative actions as shown in Fig. 2, then it is ideal to recover back to the junction state, i.e., the red state, because the states in the dead end are meaningless in achieving the goal.

Inspired by this, we record an $N$-step learning process that includes previous $N$ states, $S_N = \{s_{t-N+1}, s_{t-N+2}, \ldots, s_{t-1}, s_t\}$, and corresponding $N$ actions, $\{a_{t-N+1}, a_{t-N+2}, \ldots, a_{t-1}, a_t\}$, where $t$ is the time index, $s_t$ means the state at time $t$, and $a_t$ means the action at time $t$. The selection of $N$ should be related to the complexity of the robot, task or environment and different values of $N$ should affect the performance of RL. One of our goals is to experimentally investigate these relationships and also determine which state in these $N$ states is ideal to recover back to when a failure occurs. We assign for each of these $N$ states a recovery priority value $\theta(s)$, which indicates the priority of each state for the robot to reset to. When a state has more safe actions that do not cause failures, it should have a larger priority value. A safe action set in state $s$, $A_s(s)$, can be defined as

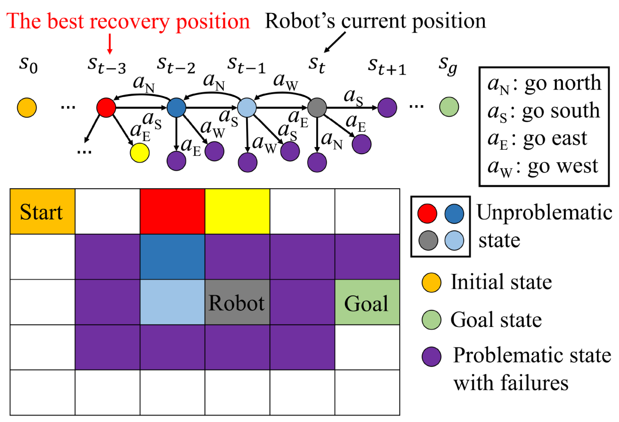

**Fig. 2 Illustration of multi-state recovery.**

$$A_s(s) \triangleq \{a \in A(s) | a \notin A_d(s)\} \tag{1}$$

where $A(s)$ means the action set containing all the actions in state $s$, and $A_d(s)$ means the dangerous action set in state $s$ and it can be updated by failures that have been detected and failures that can be predicted. In this work, we assume the failure detection as receiving a pre-defined large negative reward

$$A_d(s) \triangleq \{a | r(s,a) = r(\text{failure})\} \tag{2}$$

where $r(s,a)$ means the reward of a state-action pair $(s,a)$ and $r(\text{failure})$ means the reward of a failure.

Thus, the priority $\theta(s)$ can be calculated by

$$\theta(s) = |A_s(s)| \tag{3}$$

where $|\cdot|$ computes the size (number of elements) of a set.

If different states have the same greatest priority, then the robot will reset to the nearest state because the nearest state may be closer to the goal state. Then, the best reset state $S_r^*$ determined by the recovery strategy is

$$S_r^* = \arg\min d\left(\arg\max_{s \in S_N} \theta(s), s_t\right) \tag{4}$$

where $d\left(\arg\max_{s \in S_N} \theta(s), s_t\right)$ means the distance between an arbitrary state with the largest priority in state space $S_N$ and state $s_t$. Please note that $\arg\max_{s \in S_N} \theta(s)$ can be either one individual state or a set of multiple states.

## 3.2 Failure prevention

Although failure prevention cannot address the failures that have occurred, it can be used as another way to reduce resets from a complementary perspective of reducing potential failures.

For an unknown environment, a simple and practical method is to exactly prevent failures that have already occurred by adding corresponding unsafe actions into $A_d(s)$. As shown in Fig. 3, for example, the robot can add the action of going east into $A_d(s)$ when it collides with the obstacle for the first time. When the robot is in the grey state again next time, it will not select the action of going east to avoid the same failure. Then, the $\varepsilon$-greedy action selection strategy in traditional RL can be modified to

$$\pi(s) = \begin{cases} a = \text{rand}(A(s)) \,|\, a \notin A_d(s), & \text{if } \xi < \varepsilon; \\ \arg\max_{a \in A} Q(s_{t+1}, a) \,|\, a \notin A_d(s), & \text{otherwise} \end{cases} \tag{5}$$

where $\xi$ is a random number, $\varepsilon$ is a parameter to decide

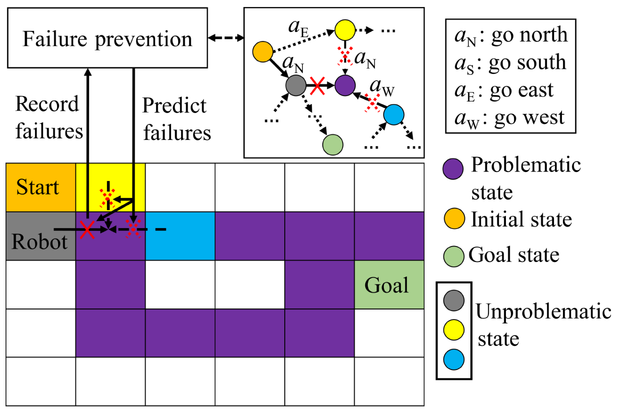

**Fig. 3 Illustration of failure prevention.**

how often an action should be selected randomly from $A_s(s)$, and $Q(s_{t+1},a)$ is the $Q$-value of the state-action pair $(s_{t+1},a)$.

If prior knowledge can be provided, we can further generate a few-shot learning model to recognize similar obstacles in the environment. As long as an obstacle is successfully recognized, the actions towards the obstacle can be inferred as unsafe and hence should be not allowed. For example, if the robot can recognize the obstacle in the state of the second row and the second column in Fig. 3, it can infer both the action of going south in the yellow state and the action of going west in the blue state as unsafe actions. Then, the failure prediction problem can be transformed to a few-shot-learning-based obstacle recognition problem.

Few-shot learning tends to have three datasets: training set, support set, and query set. The training set usually has thousands of different kinds of data with labels, and it is used to train the model with the capability of telling the difference between two common categories. The support set and the query set share the same labels, but the categories of data in them never occur in the training set. The support set provides some samples for the model to refer, while the query set is the real testing set which the model needs to classify. We call the target few-shot problem $C$-way $K$-shot if the support set contains $K$ labeled examples for each of $C$ unique classes. The size of $K$ is typically 1 or 5, that is to say, one-shot or five-shot. Our recognition is set as a 5-way 5-shot problem, and the samples in the support set never occur in the training set.

We use relation network[28] as the specific few-shot learning method in this work due to its good performance and easy implementation. We do not make any changes to the trained model and still use the similar expression to describe our problem. Specifically, our training set is based on miniImageNet[28]. The support set can be denoted as $S=\{(x_i,y_i)\}_{i=1}^{m}(m=K\times C)$, where $x_i$ and $y_i$ represent the sample and its label. We first divide the area around the obstacle into $d_0$ parts starting from east in a clockwise direction. Then the query set can be denoted as

$$Q'=\left\{\left(x_{j,s(\text{Area}_{\text{di}})},y_i\right)\right\}_{j=1}^{n},\text{di}=1,2,\ldots,d_0 \qquad (6)$$

where $\text{Area}_{\text{di}}$ represents the area of the di-th part, $s(\text{Area}_{\text{di}})$ is the initial position in $\text{Area}_{\text{di}}$ towards the obstacle at a constant distance, $x_{j,s(\text{Area}_{\text{di}})}$ means the picture captured from $s(\text{Area}_{\text{di}})$ of the $j$-th obstacle, $n$ is the number of obstacles in the environment, and $Q'$ denotes the query set.

Therefore, the recognition problem can be defined as

$$r_{i,j,s(\text{Area}_{\text{di}})}=g_\phi\left(C\left(f_\varphi(x_i),f_\varphi\left(x_{j,s(\text{Area}_{\text{di}})}\right)\right)\right),$$
$$i=1,2,3,4,5,\ \text{di}=1,2,\ldots,d_0 \qquad (7)$$

where $r_{i,j,s(\text{Area}_{\text{di}})}$ is the relation score for the relation between one query input $x_{j,s(\text{Area}_{\text{di}})}$ and training sample set examples $x_i$, $g_\phi(\cdot)$ is the relation module in the relation network, $f_\varphi(\cdot)$ is the embedding module in the relation network, and $C(\cdot)$ is the concatenation of feature maps $f_\varphi(x_i)$ and $f_\varphi(x_{j,s(\text{Area}_{\text{di}})})$ in depth.

The relation network combines the feature map of samples in the support set and query set, computes the relation scores between different categories in the support set, and then chooses the category of the biggest score to complete the classification problem. Once an obstacle is recognized, all the actions towards it should be added into $A_d(s)$, which will also update $\theta(s)$ and $\pi(s)$.

## 3.3 Algorithm summary

Our full algorithm is presented as Algorithm 1. Failure prevention is always running to predict and avoid failures and multi-state recovery is executed only when a failure occurs to save resources. In general, multi-state recovery is executed mostly in the early stage of RL. As the episode increases, RL gains more accurate knowledge about the environment and unsafe actions. This results in fewer occurrences of failures and hence fewer utilizations of the multi-state recovery.

The algorithm firstly initializes all the variables. For each episode, the algorithm updates $A_d(s)$ with the newly predicted unsafe actions and executes the modified action selection described in Eq. (5). If a

**Algorithm 1 Reset-free reinforcement learning**

```
1: Initialize all variables
2: for each episode do
3:     Update $A_{\mathrm{d}}(s)$ ⊳ failure prevention
4:     $a \leftarrow \pi(s)$ ⊳ Eq. (5)
5:     if $r(s,a) = r(\text{failure})$ then
6:         $A_{\mathrm{d}}(s) \leftarrow A_{\mathrm{d}}(s) \cup \{a\}$ ⊳ record failures
7:         Update $A_{\mathrm{s}}(s)$ ⊳ Eq. (1)
8:         Update $Q$-value
9:         $s \leftarrow S_{\mathrm{r}}^{*}$ ⊳ multi-state recovery
10:    else
11:        $s \leftarrow P(s_{t+1} | s_t, a_t)$
12:        Update $Q$-value
13:    end if
14: end for
15: Return the optimal policy
```

failure is detected (Line 5), it is recorded into the dangerous action set $A_{\mathrm{d}}(s)$ (Line 6). Based on $A_{\mathrm{d}}(s)$, the algorithm determines the ideal state to recover back to (Line 9) and continues learning in the current episode until the final training goal or the maximal step number is achieved. If no failure occurs, the algorithm determines the next state using the transition function $P(s_{t+1}|s_t,a_t)$, which indicates the probability of reaching state $s_{t+1}$ after performing action $a_t$ in state $s_t$, and then updates $Q$-values as the standard $Q$-learning does.

## 4 Result and Discussion

### 4.1 Experiment setup

In this work, we use both a maze navigation simulation and a real-world block stacking experiment to validate our algorithm.

The simulation is about robot navigation in a maze with obstacles. We set two maze environments with the maze sizes to be 4×4 and 11×11. Firstly, it is convenient for us to compare 4×4 maze results with a closely related work[2] that learns a reset policy to reduce manual resets. In addition, the 11×11 maze can be used to investigate the performances of the multi-state recovery in a more complex environment and the influences of different selections of $N$ on the performance of RL. As shown in Fig. 4, the state space of the 4×4 maze consists of 16 grids and 4 of them are obstacles. The 11×11 maze has 121 grids and 43 of them are obstacles. The robot can take four actions: go north, go south, go east, and go west. The robot starts

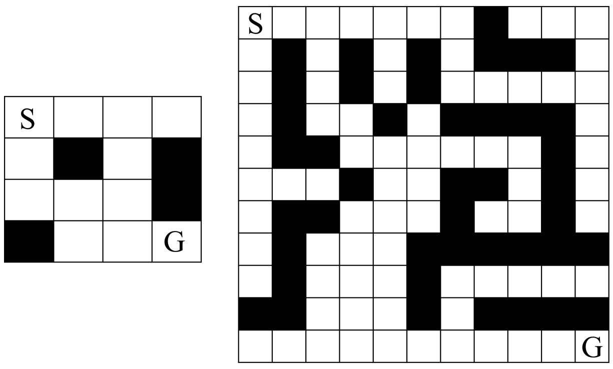

**Fig. 4 Simulation setup: 4×4 (left) and 11×11 (right) mazes. The robot is supposed to find an optimal path with the largest reward from the initial state (S) to the goal state (G). The black grids mean obstacles and the robot is not allowed to pass these obstacles. The robot can take four actions: go north, go south, go east, and go west.**

each episode from the upper-left corner of the maze and terminates if reaching the lower-right corner (goal) with a reward of 1 or hitting an obstacle with a reward of −5. For any other individual movements, the reward is −0.02. Parameters $\varepsilon$, the discount factor ($\gamma$), and the learning rate ($\alpha$) are set to be 0.1, 0.9, and 0.5, respectively.

The real-world block stacking experiment is about using a lightweight Kinova® JACO® robot arm with three fingers as shown in Fig. 5. We expect the robotic arm to stack four blocks, Block A (purple), Block B (green), Block C (orange), and Block D (blue), in the target position from their original positions. Because the grasping is not the focus of our work, we assume the original position and the target position are fixed and known so that we can guarantee a high grasping accuracy. However, the stacking order is unknown to the robot, and hence it needs reinforcement learning itself to find out. The stacking rules include: (1) Block A must be at the bottom of any other three blocks, (2) Block D must be at the top of Block C, and (3) Block D must be at the top of Block B, but not in direct contact with Block B. Violating any of these rules, the blocks will fall over, which is counted as a failure. The state space is defined as the block order at the target position. The robotic arm can take four actions: pick and place Block A, Block B, Block C, and Block D. The reward of reaching the goal (A-B-C-D) is 1 and the reward of a failure is −1. For any other individual action, the reward is 0. Parameters $\varepsilon$, $\gamma$, and $\alpha$ are set to be 0.1, 0.9, and 0.2, respectively.

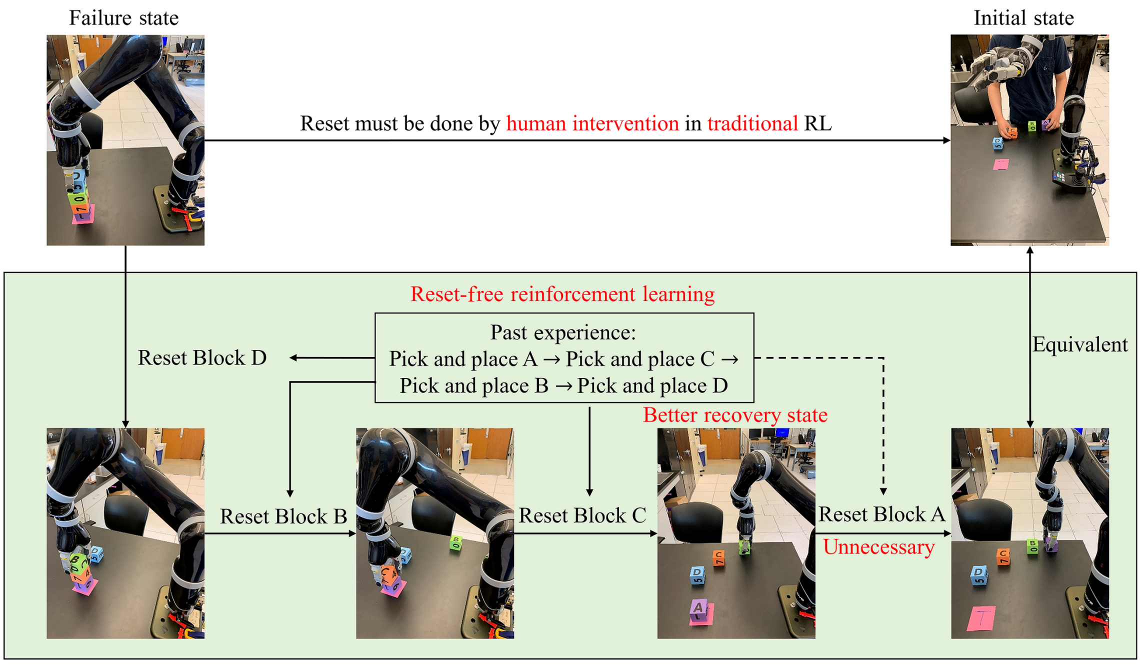
Failure state
Initial state
Reset must be done by human intervention in traditional RL
Reset-free reinforcement learning
Past experience:
Pick and place A → Pick and place C →
Pick and place B → Pick and place D
Reset Block D
Equivalent
Better recovery state
Reset Block B
Reset Block C
Reset Block A
Unnecessary

**Fig. 5 Capability of self-recovering back to any state. Human intervention is required in traditional reinforcement learning to reset a failure state to the initial state. In contrast, our self-recoverable reset strategy can recover back to any states with reversely executing the safe actions recorded in the experience buffer. With the multi-state recovery, our method could decide a better state to recover back to. In the situation that our method determines to recover back to the initial state, it is equivalent to the manual reset.**

The calculation of the number of resets includes two parts. The first one is adding one reset if the robot cannot achieve the goal state within the limited steps in an episode. We do not count the transition from the goal state in an episode to the initial state in the next episode as a reset because we focus on the failure-induced resets in this work. The other part comes from the occurrence of failures that are defined as collisions between the robot and obstacles. If not using multi-state recovery, a reset will be needed whenever a failure happens. In order to provide reliable results, the simulations and the real-world experiments in this work have been conducted 50 and 20 times, respectively, to compute the average.

For a comprehensive validation, we firstly assume the maze environment is totally unknown, which means failure prevention can only use the simple failure recording method. In this setting, we will compare our algorithm with related work and investigate the parameters of multi-state recovery as well as the influences of multi-state recovery and failure prevention on RFRL. Then, we will assume the maze environment is known to validate the more complex failure prevention based on few-shot learning. The dynamics is also changed from deterministic to stochastic to validate whether RFRL still works. Lastly, we evaluate the performance of RFRL on real-world experiments to see if it is consistent with simulations. For all validations, multi-state recovery and failure prevention are also used as independent methods. When multi-state recovery works independently, it uses the $\varepsilon$-greedy action selection strategy as in traditional RL. When failure prevention works independently, the learning episode terminates and a reset must be needed once a failure occurs.

## 4.2 Result

For a convenient expression and comparison, multi-state recovery and failure prevention are abbreviated to MSR and FP, respectively.

**(1) Comparison with related reset policy**

Table 1 shows that MSR, FP, and RFRL can all reduce much more resets than the reset policy proposed by Eysenbach et al.[2] We use the best result of their reset policy with the early abort threshold $q_{\min} = 0.4$ for comparison. For our methods, $N$ is chosen to be 3 and the maximum step per episode is set as 200 because the 4×4 maze is relatively simple.

Although both Eysenbach's reset policy and our FP method reduce the resets by reducing potential failures,

**Table 1 Comparisons with previous results. Any of our three methods outperforms the reset policy proposed by Eysenbach et al.[2] in terms of both the number of resets and the number of total steps.**

| Method | Number of resets (mean) | Number of total steps (mean) |
|---|---|---|
| Reset policy[2] | 55 | 6200 |
| FP | 9 | 1720 |
| MSR | 0 | 1738 |
| RFRL | 0 | 1694 |

our FP method is still better because we can exactly avoid recorded failures that have occurred while Eysenbach uses a threshold to partially abort some dangerous actions. Because MSR gives RL the ability to recover from failures, both MSR and RFRL can completely avoid resets. In addition, our method requires much fewer steps to finish the training, which can be regarded as more efficient.

**(2) MSR vs. FP in RFRL**

As discussed before, either MSR or FP can individually reduce the number of resets. In order to investigate the effects in detail, we calculate the number of resets, number of total steps to converge, and number of episodes to converge in Table 2. For the simple 4×4 maze and the complex 11×11 maze, both MSR and FP can significantly reduce the number of resets compared to standard $Q$-learning. While FP can only reduce resets, MSR can guarantee no resets in the simple maze and a mostly zero reset in the complex maze. This is because MSR can substantially eliminate resets through recovering from failures. In contrast, FP only avoids resets by preventing failures, which can be easily restricted with a necessity of accurate knowledge about the environment. Combining MSR and FP to RFRL, the performance improves to the best. Both FP and MSR can find the optimal path with fewer episodes and total steps than the standard $Q$-learning. However, MSR is more efficient, especially in the complex maze.

**Table 2 Full comparisons among the standard *Q*-learning, *Q*-learning algorithm with FP, *Q*-learning with MSR, and RFRL. Either FP or MSR can outperform the standard *Q*-learning, although the combination of FP and MSR to RFRL has the best performance in all the four methods. MSR can almost guarantee a zero reset even in a complex 11×11 maze because it avoids the resets from the perspective of fixing failures rather than just avoiding failures.**

| Method | Number of resets (mean) | Number of episodes (mean) | Number of total steps (mean) |
|---|---|---|---|
| Standard-4×4 | 33.5 | 243.0 | 1796 |
| FP-4×4 | 9.0 | 217.6 | 1720 |
| MSR-4×4 | 0.0 | 211.8 | 1738 |
| RFRL-4×4 | 0.0 | 207.8 | 1694 |
| Standard-11×11 | 345.5 | 524.0 | 10 610 |
| FP-11×11 | 95.1 | 327.6 | 10 658 |
| MSR-11×11 | 0.9 | 234.2 | 9887 |
| RFRL-11×11 | 0.7 | 235.0 | 9709 |

**(3) Will MSR cause more failures to recover from a failure?**

One important concern for MSR is whether more failures can be caused during the recovery from a specific failure to previous states.

Firstly, Table 3 shows that MSR does not really have such a concern. Although the number of total failures in MSR is close to that in the standard $Q$-learning, it is still less than the standard $Q$-learning. This is because MSR cannot change the essence of RL, i.e., trial and error. MSR only changes the timing of exploring such failures, which is exploring more failures than the standard $Q$-learning in the early episodes. This expedites the learning process with fewer episodes, which can avoid the failures in the unnecessary episodes. Because most failures have been encountered in such episodes, $Q$-learning may also avoid them based on the $Q$-values. However, the random exploration strategy in the standard $Q$-learning can still cause a few failures, which explains the small failure difference between MSR and the standard $Q$-learning.

Secondly, FP has far fewer failures than MSR because all the random explorations in FP are strictly restricted to safe actions. In other words, the same

**Table 3 Failure comparisons between the standard *Q*-learning and our method. While MSR only reduces the failures in a small amount, FP can significantly reduce the failures because it can strictly restrict the random explorations in the *Q*-learning to safe actions. However, the failures in MSR do not result in resets while the failures in the standard *Q*-learning and FP must require corresponding resets.**

| Method | Number of total failures (mean) |
|---|---|
| Standard-4×4 | 33.5 |
| FP-4×4 | 9.0 |
| MSR-4×4 | 32.1 |
| RFRL-4×4 | 10.0 |
| Standard-11×11 | 345.5 |
| FP-11×11 | 95.1 |
| MSR-11×11 | 333.2 |
| RFRL-11×11 | 114.7 |

mistake is not allowed to be made again in FP. Although the number of failures caused by normal random explorations can be small, multiplication by the number of episodes can be significant; this explains the large failure difference between FP/RFRL and standard *Q*-learning/MSR.

Thirdly, although MSR has a certain number of failures, these failures do not result in resets because they can be recovered by RL itself. For example, the MSR explores an average of 32.1 failures without any reset as indicated in Table 2. In contrast, the number of failures is equal to the number of resets for the standard *Q*-learning and FP because a reset is required whenever a failure occurs in these two methods.

**(4) *N* and maximum steps per episode in MSR**

As discussed in the multi-state recovery strategy, the selection of *N* and maximum steps per episode is important in MSR. We empirically allow the variations of *N* and maximum steps per episode to be 1 to 7 with an increment of 1 and 100 to 1000 with an increment of 100, respectively.

Figure 6 shows the relationship between the number of resets and the number of previous safe states *N* that has been recorded. *N* can be also interpreted as recovery limit, i.e., the maximum step the robot can go back. When the environment is simple (the 4×4 maze), MSR can guarantee no resets. However, when the environment is more complex (the 11×11 maze), a larger recovery limit *N* could be better especially when allowing enough maximum steps in one episode. The limited maximum step per episode such as 400 for the 11×11 maze could be too small for MSR to be fully functional because recovering from failures needs to cost steps. Thus, when the maximum step per episode is small (400), a lower recovery limit ($N$ = 1) outperforms other *N*'s. As the maximum step per episode increases to 600 or 800, however, $N = 2$ or $N = 3$ actually has the best performance. The differences between various *N*'s should be much more significant when the environment becomes highly complex.

Figure 7 shows the relationship between the number of resets and maximum steps per episode. A general conclusion is that the number of resets reduces as the maximum step per episode increases, which provides enough time for MSR to recover from failures. In addition, Fig. 7 also proves that a larger *N* has better performances when the maximum step per episode is larger.

**(5) Failure prevention with few-shot learning**

In order to use the few-shot learning for failure prevention, we assume that some of obstacles are known and some of them belong to the known category but have not been seen before. The parameter $d_0$ in Eq. (6) is set as 4, which means all the pictures are captured from north, south, east, and west. As shown in Fig. 8, our query set consists of 5 categories which contain chairs, kettles, cartons, basins, and baskets, and each category has 8 pictures. As 4 pictures describe an obstacle, 40 pictures actually indicate 10 obstacles. For

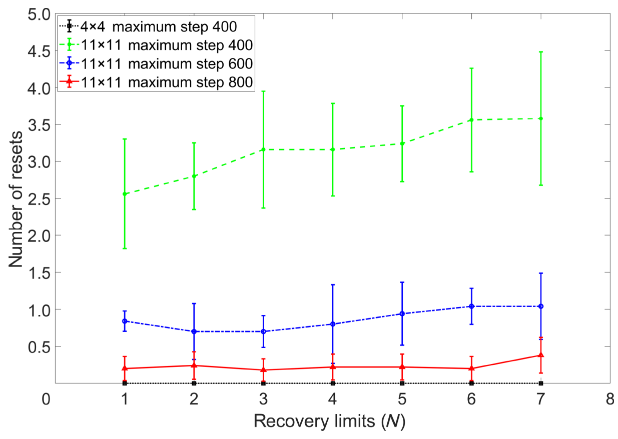

**Fig. 6 Relationship between the number of resets and recovery limits *N*. When the environment is simple (a 4×4 maze), even only recovering back to the latest state can guarantee no resets. However, when the environment is more complex (a 11×11 maze), a larger recovery limit *N* (e.g., 3) could be more effective, especially given enough maximum steps per episode.**

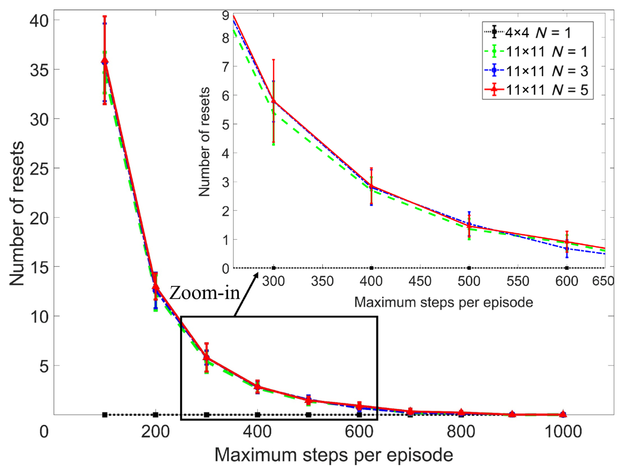

**Fig. 7 Relationship between the number of resets and maximum steps per episode. The number of resets reduces as the maximum steps per episode increase. When maximum steps per episode remain large, a larger *N* can be better in reducing the number of resets.**

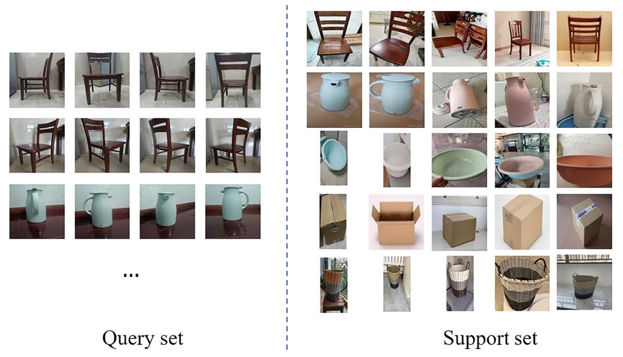

**Fig. 8 Query set and support set for the maze environment.**

the 4×4 maze, we randomly choose 4 from the 10 obstacles. For the 11×11 maze, obstacles are allowed to be repetitive, but the number of duplicates should be less than 5 to guarantee the existence of each obstacle in the maze.

Table 4 shows more results of failure prevention based on few-shot learning. Because standard *Q*-learning and MSR are not affected by FP, their results remain the same in Table 2 and hence are not listed. As shown in Table 4, failure prevention based on few-shot learning can further reduce both the number of resets and the number of failures compared with Table 2. Due to prior knowledge, FP can accurately recognize all the obstacles in the 4×4 maze and hence excludes all the unsafe actions, resulting in a zero failure and a zero reset. For the 11×11 maze, FP can reduce the number of failures by 49%, and RFRL can guarantee a zero reset. It should be also noted that the reduction of episodes and total steps in 11×11 maze is more obvious than that in 4×4 maze, which indicates the fusion of external knowledge can be greatly useful in complex environments.

**(6) Stochastic dynamics**

All of the previous simulations are based on deterministic dynamics. However, the practical dynamics can be stochastic, which means the result of an action is not always as expected. For example, if the robot executes the action of going north, it may actually reach the state in the west or in the east instead of the expected state in the north. In the maze simulation, we assume the probability of reaching the expected state is 0.8 and the probabilities of reaching the states in the left or the right are both 0.1.

As shown in Table 5, RFRL still works in stochastic dynamics because the number of resets has only a very small increment, which is still close to zero. However, the increments of failures, episodes, and total steps are obvious because a safe action could also lead to a failure in stochastic dynamics. In all the four methods, RFRL still performs the best, which is consistent with previous results. It is also interesting to see that the increments of failures, episodes, and total steps in the 11×11 maze are larger than those in the 4×4 maze. For instance, the number of failures using RFRL increases by 8% in the 4×4 maze, but increases by 13% in the 11×11 maze. The possible reason is that only one path

**Table 4 Results of failure prevention based on few-shot learning. The use of prior knowledge can avoid more failures and require fewer resets.**

| Method | Number of resets (mean) | Number of total failures (mean) | Number of episodes (mean) | Number of total steps (mean) |
|---|---|---|---|---|
| FP-4×4 | 0.0 | 0.0 | 208.5 | 1650 |
| RFRL-4×4 | 0.0 | 0.0 | 200.7 | 1624 |
| FP-11×11 | 48.5 | 48.5 | 207.3 | 5527 |
| RFRL-11×11 | 0.0 | 56.9 | 150.7 | 5122 |

**Table 5 Results of stochastic dynamics. The performance of each algorithm is worse compared with deterministic dynamics, but our algorithm still works and performs better than standard *Q*-learning. RFRL still has the best performance.**

| Method | Number of resets (mean) | Number of total failures (mean) | Number of episodes (mean) | Number of total steps (mean) |
|---|---|---|---|---|
| Standard-4×4 | 38.8 | 38.8 | 284.6 | 2156 |
| FP-4×4 | 10.0 | 10.0 | 261.2 | 2068 |
| MSR-4×4 | 0.0 | 37.8 | 256.3 | 2059 |
| RFRL-4×4 | 0.0 | 10.8 | 248.3 | 2030 |
| Standard-11×11 | 413.7 | 413.7 | 620.9 | 12 584 |
| FP-11×11 | 108.4 | 108.4 | 376.6 | 12 316 |
| MSR-11×11 | 1.0 | 382.1 | 265.2 | 11 580 |
| RFRL-11×11 | 0.9 | 129.6 | 265.5 | 11 017 |

exists with more obstacles in the 11×11 maze. As the maze size further increases, the number of failures may keep increasing too.

**(7) Block stacking results**

As shown in Table 6, the block stacking results are consistent with the maze navigation simulation results. RFRL still performs the best with the fewest number of resets, the fewest number of episodes, and the fewest number of total steps. The number of resets is zero if MSR is used. FP can also reduce the number of resets, but it still has 11.6 resets in average.

As shown in Fig. 5, our method firstly provides the robot with a capability of self-recovering to any state in its stored experience. This completely avoids human intervention that is required in traditional reinforcement learning. Even if the robot does not know which state it should recover back to, it can at least recover back to the initial state, which is equivalent to the result of a manual reset. If the robot has more information learned from failures or directly from an external guidance, it can decide a better state to recover back to.

In order to investigate the influence of $N$ and the maximum number of steps per episode on RFRL, we empirically allow the variation of $N$ to be 0 to 4 with an increment of 1. $N = 0$ means no recovery at all, while $N = 4$ means that four maximum steps can be recovered but not all the four steps should be recovered. In order to investigate the influence of the maximum number of steps per episode on RFRL, we also empirically allow the maximum number of steps per episode to be 10, 20, 50, and 100, respectively.

Table 7 shows that the number of manual resets increases as the recovery limit $N$ increases. This is because when the recovery limit is low, there could be failures that cannot be recovered. For example, when only one step ($N = 1$) is allowed to be recovered, the failure (A-C-D) can be only recovered back to the state (A-C), which can never succeed since another possible stacking order (A-C-B-D) from this state is also a failure and cannot be recovered either. When $N$ increases to 2, the failure (A-C-D) can be recovered but the failure (A-C-B-D) still cannot be recovered. Increasing $N$ to 3 and 4 can recover any failures, but $N = 4$ is actually worse than $N = 3$ due to more training time. This means that a larger recovery limit could be more than sufficient and hence result in unnecessary recovery, which costs more training time. Because each episode automatically terminates when it successfully achieves the goal state, more maximum steps per episode do not bring negative influences as a larger recovery limit does. However, if the robot gets stuck in

**Table 6 Full comparisons among the block stacking results using standard *Q*-learning, FP, MSR, and RFRL. Either FP or MSR can individually outperform the standard *Q*-learning, although the combination of FP and MSR to RFRL has the best performance in all the four methods. Both MSR and RFRL can guarantee a zero reset.**

| Method for block stacking | Number of resets (mean) | Number of failures (mean) | Number of episodes (mean) | Number of total steps (mean) |
|---|---|---|---|---|
| Standard | 41.2 | 41.2 | 157.0 | 3824 |
| FP | 11.6 | 11.6 | 133.2 | 2516 |
| MSR ($N = 3$) | 0.0 | 39.8 | 124.3 | 2681 |
| RFRL | 0.0 | 12.2 | 120.7 | 2294 |

**Table 7 Number of manual resets and total training time with respect to different recovery limits and different maximum steps per episode.**

| Parameter | | Number of manual resets (mean) | Total training time (mean) |
|---|---|---|---|
| Different recovery limits $N$ (maximum steps per episode = 50) | 0 | 41.2 | 5.5 h |
| | 1 | 22.5 | 4.8 h |
| | 2 | 10.8 | 4.4 h |
| | 3 | 0.0 | 3.6 h |
| | 4 | 0.0 | 3.9 h |
| Different maximum steps per episode (recovery limits $N = 3$) | 10 | 28.4 | 5.2 h |
| | 20 | 11.1 | 4.5 h |
| | 50 | 0.0 | 3.6 h |
| | 100 | 0.0 | 3.6 h |

a failure such as when the recovery limit is 1, more maximum steps per episode could cause the robot to keep making mistakes (e.g., keep stacking D in the state A-C-B) until the maximum step is reached, which costs a number of unnecessary training time.

### 4.3 Discussion

The general goal of our work is to assist RL-based robots to be truly autonomous in practical applications by solving an unavoidable problem: Robots require manual resets whenever encountering failures during the learning. While this can be solved by reducing failures as done in previous work, we, more importantly, propose a multi-state recovery strategy to fix the failures. This is similar to the advice for many complex systems: "We should not wonder if some mishap may happen, but rather ask what one will do about it when it occurs"[29]. We have also demonstrated that the combination of multi-state recovery and failure prevention can achieve better performances than either individual method by guaranteeing no resets in simple scenarios.

Within the self-recovery from failures, we specifically propose a multi-state recovery strategy by investigating which previous state is ideal to recover back to when a failure occurs. This is actually inspired by human's self-recovery from failures. For example, when humans experience a failure in a jigsaw puzzle, they do not really go back to start from the first piece or just revert the last piece, it is usual to take out several recently inserted pieces and restart from such a configuration. We achieve this by using memory to record some past experiences such as a certain-length sequence of state-action pairs in RL. This could cost more resources in robots, but the increment of resources is believed to be affordable in practical systems because the amount of data is much smaller if compared to the memory prepared for RL. The storage of the matrix about unsafe state-action pairs can be another cost of memory, but this cost can be greatly reduced using compressed storage because the matrix is usually highly sparse.

The failure discussed in this work is assumed to be recoverable, which means it will not cause fatal damages to the robot. For fatal failures, manual resets are certainly needed. In addition, although we focus on the failure-induced resets because the occurrence of failures is a primary reason for manual resets in practice, it cannot be denied that manual resets can also be caused by other reasons. A representative example is the transition from the goal state in one episode to the initial state in the next episode. This could require significant resets if RL requires massive learning episodes, especially for highly complex tasks. However, unlike the failure-induced resets, this kind of resets does not necessarily need human intervention because it can be the same for a specific task. Robots may use pre-defined plans to autonomously go back to the initial state.

In this work, we have only discussed the performance of our method in a static environment. A more challenging case can be a dynamic environment. While our work may still work if the time-varying change is small, it is better to design an advanced algorithm to completely address dynamic changes. Unlike the stochastic dynamics, it is more difficult for multi-state recovery and RFRL to work in the dynamic environment because the past experiences can be no longer accurate and cannot be inferred or reconstructed without any other assistance. One possible solution is to estimate a dynamic environment model[30, 31] and then predict the changes in the past experience using model predictive control[32]. The estimation of the state-transition probability of the environment can be achieved by Bayes inference with forgetting effect[33]. We may also generate a map about the environment through simultaneous localization and mapping methods such as gmapping[34] and multi-robot cooperative map fusion[35–37].

## 5 Conclusion

In this work, we propose a reset-free reinforcement learning algorithm to avoid manual resets in the practical deployment of RL on autonomous robots. It includes multi-state recovery and failure prevention. Instead of simply abandoning the failed episode without further learning within it, the multi-state recovery focuses on recovering from failures so that RL can continue learning to succeed in finishing the training goal in the failed episode. The failure prevention predicts and avoids potential failures according to previous experiences. Although these two parts can individually reduce resets, the combination of them is experimentally demonstrated to achieve fewer resets with using fewer episodes.

## Acknowledgment

This work was supported by the National Natural

Science Foundation of China (No. 61876024), partly by the Higher Education Colleges in Jiangsu Province (No. 21KJA510003) and the Suzhou Municipal Science and Technology Plan Project (Nos. SYG202351 and SYG202129).

## References

[1] W. Han, S. Levine, and P. Abbeel, Learning compound multi-step controllers under unknown dynamics, in *Proc. 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS*), Hamburg, Germany, 2015, pp. 6435–6442.

[2] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, Leave no Trace: Learning to reset for safe and autonomous reinforcement learning, presented at 6th International Conference on Learning Representations (ICLR), Vancouver, Canada, 2017.

[3] A. Gupta, J. Yu, T. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine, Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention, in *Proc. 2021 International Conference on Robotics and Automation* (*ICRA*), Xi'an, China, 2021, pp. 6664–6671.

[4] V. Verma, G. Gordon, R. Simmons, and S. Thrun, Real-time fault diagnosis [robot fault diagnosis], *IEEE Robot. Autom. Mag.*, vol. 11, no. 2, pp. 56–66, 2004.

[5] S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar, Generation of whole-body optimal dynamic multi-contact motions, *Int. J. Robot. Res.*, vol. 32, nos. 9&10, pp. 1104–1119, 2013.

[6] V. Vonásek, S. Neumann, D. Oertel, and H. Wörn, Online motion planning for failure recovery of modular robotic systems, in *Proc. 2015 IEEE International Conference on Robotics and Automation* (*ICRA*), Seattle, WA, USA, 2015, pp. 1905–1910.

[7] K. Chatzilygeroudis, V. Vassiliades, and J. B. Mouret, Reset-free trial-and-error learning for robot damage recovery, *Robot. Auton. Syst.*, vol. 100, pp. 236–250, 2018.

[8] J. Kober, J. A. Bagnell, and J. Peters, Reinforcement learning in robotics: A survey, *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.

[9] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation, in *Proc. 2018 IEEE International Conference on Robotics and Automation* (*ICRA*), Brisbane, Australia, 2018, pp. 5129–5136.

[10] K. Zhu and T. Zhang, Deep reinforcement learning based mobile robot navigation: A review, *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

[11] L. Wang, Z. Pan, and J. Wang, A review of reinforcement learning based intelligent optimization for manufacturing scheduling, *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 257–270, 2021.

[12] X. Wang, L. Wang, C. Dong, H. Ren, and K. Xing, Reinforcement Learning-Based Dynamic Order Recommendation for On-Demand Food Delivery, *Tsinghua Science and Technology*, vol. 29, no. 2, pp. 356–367, 2024.

[13] M. Deisenroth and C. E. Rasmussen, PILCO: A model-based and data-efficient approach to policy search, in *Proc. 28th International Conference on Machine Learning* (*ICML*), Bellevue, WA, USA, 2011, pp. 465–472.

[14] J. Peters and S. Schaal, Policy gradient methods for robotics, in *Proc. 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS*), Beijing, China, 2006, pp. 2219–2225.

[15] J. Garcıa and F. Fernandez, A comprehensive survey on safe reinforcement learning, *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1437–1480, 2015.

[16] P. Geibel and F. Wysotzki, Risk-sensitive reinforcement learning applied to control under constraints, *J. Artif. Intell. Res.*, vol. 24, pp. 81–108, 2005.

[17] Y. Kadota, M. Kurano, and M. Yasuda, Discounted Markov decision processes with utility constraints, *Comput. Math. Appl.*, vol. 51, no. 2, pp. 279–284, 2006.

[18] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, Nonparametric return distribution approximation for reinforcement learning, presented at the 27th International Conference on Machine Learning (ICML), Haifa, Israel, 2010.

[19] T. M. Moldovan and P. Abbeel, Safe exploration in Markov decision processes, presented at the 29th International Conference on International Conference on Machine Learning (ICML), Edinburgh, UK, 2012.

[20] K. Driessens and S. Džeroski, Integrating guidance into relational reinforcement learning, *Mach. Learn.*, vol. 57, no. 3, pp. 271–304, 2004.

[21] P. Abbeel, A. Coates, and A. Y. Ng, Autonomous helicopter aerobatics through apprenticeship learning, *Int. J. Robot. Res.*, vol. 29, no. 13, pp. 1608–1639, 2010.

[22] J. Tang, A. Singh, N. Goehausen, and P. Abbeel, Parameterized maneuver learning for autonomous helicopter flight, in *Proc. 2010 IEEE International Conference on Robotics and Automation* (*ICRA*), Anchorage, AK, USA, 2010, pp. 1142–1148.

[23] C. Gehring and D. Precup, Smart exploration in reinforcement learning using absolute temporal difference errors, presented at the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), St. Paul, MN, USA, 2013.

[24] J. Morimoto and K. Doya, Robust reinforcement learning, *Neural Comput.*, vol. 17, no. 2, pp. 335–359, 2005.

[25] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, Robust adversarial reinforcement learning, in *Proc. 34th International Conference on Machine Learning* (*ICML*), Sydney, Australia, 2017, pp. 2817–2826.

[26] Y. Li, Y. Tian, E. Tong, W. Niu, Y. Xiang, T. Chen, Y. Wu, and J. Liu, Curricular robust reinforcement learning via GAN-based perturbation through continuously scheduled task sequence, *Tsinghua Science and Technology*, vol. 28, no. 1, pp. 27–38, 2023.

[27] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA: MIT press, 1998.

[28] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. Torr, and T. Hospedales, Learning to compare: Relation network for few-shot learning, in *Proc. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), Salt Lake City, UT, USA, 2018, pp. 1199–1208.

[29] F. Corbato, On building systems that will fail, *Commun. ACM*, vol. 34, no. 9, pp. 72–81, 1991.

[30] F. Cherni, M. Boujelben, L. Jaiem, Y. Boutereaa, C. Rekik, and N. Derbel, Autonomous mobile robot navigation based on an integrated environment

representation designed in dynamic environments, *Int. J. Autom. Control*, vol. 11, no. 1, pp. 35–53, 2017.

[31] C. Hu, R. Qiao, Z. Zhang, X. Yan, and M. Li, Dynamic scheduling algorithm based on evolutionary reinforcement learning for sudden contaminant events under uncertain environment, *Complex Systems Modeling and Simulation*, vol. 2, no. 3, pp. 213–223, 2022.

[32] J. Xin, Y. Qu, F. Zhang, and R. Negenborn, Distributed model predictive contouring control for real-time multi-robot motion planning, *Complex System Modeling and Simulation*, vol. 2, no. 4, pp. 273–287, 2022.

[33] S. Ishii, W. Yoshida, and J. Yoshimoto, Control of exploitation-exploration meta-parameter in reinforcement learning, *Neural Netw.*, vol. 15, nos. 4–6, pp. 665–687, 2002.

[34] K. Zhang and L. Ning, Hybrid navigation method for multiple robots facing dynamic obstacles, *Tsinghua Science and Technology*, vol. 27, no. 6, pp. 894–901, 2022.

[35] S. Sun and B. Xu. Online map fusion system based on sparse point-cloud, *Int. J. Autom. Control*, vol. 15, nos. 4&5, pp. 585–610, 2021.

[36] Z. Li, B. Xu, D. Wu, K. Zhao, S. Chen, M. Lu, and J. Cong, A YOLO-GGCNN based grasping framework for mobile robots in unknown Environments, *Expert Syst. Appl.*, vol. 225, 2023.

[37] H. Lu, S. Yang, M. Zhao, and S. Cheng, Multi-robot indoor environment map building based on multi-stage optimization method, *Complex System Modeling and Simulation*, vol. 1, no. 2, pp. 145–161, 2021.

**Xu Zhou** received the BEng degree in automation from Nanjing University of Information Science and Technology, Nanjing, China, in 2011, the MEng degree in control theory and control engineering from Nanjing University of Science and Technology, China, in 2014, and the PhD degree in mechanical engineering from Colorado School of Mines in Golden, USA, in 2019. He is currently an assistant professor at School of Mechanical Engineering, Changshu Institute of Technology, Changshu, China. His current research interests include intelligent robot control, reinforcement learning, and knowledge-based systems.

**Zhengqiang Jiang** received the PhD degree from The University of Western Australia, Perth, Australia, in 2014. After his PhD graduation, he worked at University of Houston, Changshu Institute of Technology, University of Technology Sydney, and Macquarie University. He is currently at Discipline of Medical Imaging, Faculty of Medicine and Health, The University of Sydney, Australia. His research interests include artificial intelligence, object detection, object tracking, medical imaging, and image perception, with papers published in *IEEE Transactions on Image Processing* and *Journal of the Optical Society of America A*.

**Brett Nener** received the BEng and PhD degrees from The University of Western Australia, Australia, in 1977 and 1987, respectively, and the MSc degree from The University of Tokyo, Japan, in 1980. He has been a visiting professor with the U.S. Navy Space and Naval Warfare Center, San Diego, CA, USA, University of California, Santa Barbara, and Japanese National Institute of Information and Communications Technology (NICT). He is currently a professor at Department of Electrical, Electronic and Computer Engineering, The University of Western Australia, where he was the head of the department from 2008 to 2014.

**Benlian Xu** received the PhD degree in control science and engineering from Nanjing University of Science and Technology, Nanjing, China, in 2006. As a visiting fellow, he was invited to join the research project on biomedical image analysis in The University of Melbourne, The University of Western Australia, and The Australian National University in 2009, 2012, and 2018, respectively. He is currently a professor at School of Electronic and Information Engineering, Suzhou University of Science and Technology, Suzhou, China. His current research interests focus on multi-object tracking, swarm intelligence, and simultaneous localization and mapping.

**Jun Li** received the BEng degree in automatic control from East China Institute of Technology, Nanjing, China, in 1991, the MEng degree in control theory and control engineering from East China Institute of Technology, Nanjing, China, in 1994, and the PhD degree in control science and engineering from Nanjing University of Science and Technology, Nanjing, China, in 1998. He is currently a professor at School of Automation, Nanjing University of Science and Technology, Nanjing, China. His current research interests include intelligent control, robot learning, and computer vision.