Failure-aware Policy Learning for Self-assessable Robotics Tasks

Kechun Xu, Runjian Chen, Shuqi Zhao, Zizhang Li, Hongxiang Yu, Ci Chen, Yue Wang, Rong Xiong

***Abstract*— Self-assessment rules play an essential role in safe and effective real-world robotic applications, which verify the feasibility of the selected action before actual execution. But how to utilize the self-assessment results to re-choose actions remains a challenge. Previous methods eliminate the selected action evaluated as failed by the self-assessment rules, and re-choose one with the next-highest affordance (*i.e.* process-of-elimination strategy [1]), which ignores the dependency between the self-assessment results and the remaining untried actions. However, this dependency is important since the previous failures might help trim the remaining over-estimated actions. In this paper, we set to investigate this dependency by learning a failure-aware policy. We propose two architectures for the failure-aware policy by representing the self-assessment results of previous failures as the variable state, and leveraging recurrent neural networks to implicitly memorize the previous failures. Experiments conducted on three tasks demonstrate that our method can achieve better performances with higher task success rates by less trials. Moreover, when the actions are correlated, learning a failure-aware policy can achieve better performance than the process-of-elimination strategy.**

I. Introduction

A crucial problem for robotic applications in real world is how to promise action safety and effectiveness, especially for the applications of robot learning policies in unseen testing scenarios. A common way to deal with this problem is to utilize pre-defined self-assessment rules, which verify the feasibility of the selected actions before actual robot execution. For example, most of works in autonomous driving [2], [3], [4], [5], [6], [7] utilize the pre-built global map to forecast potential collision with the selected action. Once the selected action fails to pass the self-assessment tests, the learned policy has to re-choose an action. However, since there is no actual action execution, the observation stays invariant. Thus, the same action will be re-chosen by the learned policy and fail again, which raises a problem: how to re-choose actions among the remaining untried ones?

Considering that the self-assessment results enable the re-decision of a sequence of actions, in this paper, we formally state this re-choosing process as a sequential decision making problem under an invariant observation. Specifically, as shown in Fig. 1(a), given an observation o, a learned policy $\pi_0(o)$ generates the initial action a_0, followed by a self-assessment module **SA** to indicate success or failure. If failed, the failure-aware policy π_{FA} is activated for re-choosing the action. Conditioned on the observation o, $\pi_{\text{FA}}(s_t|o)$ takes the self-assessment results of the previous actions as state s_t to generate a new action a_t at step t until getting successful feedback from **SA**. Given the action affordance distribution predicted from π_0, one intuitive way to design π_{FA}, which is shown in Fig. 1(b), is to choose the action with the next-highest affordance if the previously selected action is evaluated as unqualified, and so forth (*i.e.* process-of-elimination strategy [1], [2], [3], [4], [5], [6], [7]). However, there raises a further question of whether it is an optimal failure-aware policy. In other words, *under invariant observation, does the equality hold between the action with the second-highest affordance and the action with the highest affordance conditioned on previous failures?* The process-of-elimination strategy gives a positive answer to this question, which means previous action failures cause no influence on the affordance distribution of the remaining untried actions. However, we argue that the previous failure is an important prior for the action re-choosing. Therefore, the equality might not always hold.

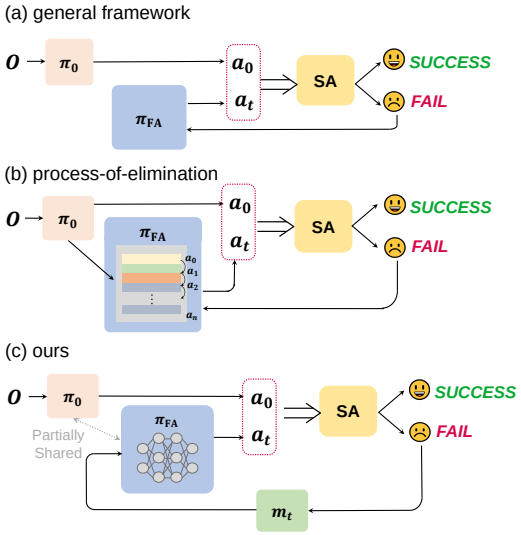


Fig. 1. (a) The general framework to utilize the self-assessment module. A learned policy π_0 takes as input the observation o, and selects the initial action a_0. Then a self-assessment module **SA** evaluates the selected action. If failed, a failure-aware policy π_{FA} will re-choose the action a_t until getting successful feedback. (b) A common pipeline (process-of-elimination [1]) to design π_{FA}, which re-chooses action a_t in a sorting way using the action affordance map generated from π_0. (c) Our pipeline, which constructs a failure memory representation m_t and uses a learning-based π_{FA}.

In this paper, we set to investigate the dependency between the self-assessment results and the remaining untried actions by learning the failure-aware policy π_{FA} (Fig. 1(c)). We define self-assessable robotics tasks as those where the robot can evaluate itself by some self-assessment rules before actual action execution. Our key insight is to integrate the self-assessment results during the observation-invariant process

This work was supported in part by the National Key R & D Program of China under Grant 2021ZD0114500. Kechun Xu, Shuqi Zhao, Zizhang Li, Hongxiang Yu, Ci Chen, Yue Wang, Rong Xiong are with Zhejiang University, Hangzhou, China. Runjian Chen is with The University of Hong Kong. Corresponding author,wangyue@iipc.zju.edu.cn.

into the training of π_{FA}. We represent the results of the previous failure verified by the self-assessment module as m_t, which serves as the variable state s_t of π_{FA}. Also, Recurrent Neural Networks [8], [9] are helpful for the implicit representation to memorize the previous failures. Based on these points, we propose two architectures for the failure-aware policy. One tends to explicitly degenerate actions similar to failed ones, while another uses recurrent network to implicitly represent failure memory of the trial sequence. Experiments conducted on three tasks demonstrate that our method can achieve better performances with higher task success rates by less trials. Moreover, we find that when the actions are correlated, learning a failure-aware policy can achieve better performance than the process-of-elimination strategy. To summarize, our contributions are as follows:

- Our main contribution is to provide a learning-based perspective to utilize self-assessment results to learn a failure-aware policy for self-assessable robotics tasks.
- We propose two effective architectures for the failure-aware policy. One tends to degenerate actions similar to failed ones, while another uses recurrent network to implicitly represent failure memory of the trial sequence.
- We evaluate our method with three typical self-assessable robotics tasks, including sequential image classification, object reorientation and localization. Both simulated and real-world experiments validate the effectiveness of our policy, and the two architectures present different advantages according to the task properties. Moreover, when the actions are correlated, learning a failure-aware policy can achieve better performance than the process-of-elimination strategy.

II. Related Works

Robotic Self-assessment. Recently, robotic self-assessment has become a topic of interest in human-robot interaction. [10] highlights the importance of online Competence Assessment (CA) for safe real-world operation of robots. [11] further extends the term to Proficiency Self-Assessment (PSA), which shows the ability of a robot to predict, estimate, or measure its performance given a context or environment before action execution. Actually, this term can be extended to all robotics tasks, and a self-assessable robotics task means that the robot has some PSA metrics to evaluate its performance. For example, lots of works discard unsafe selected actions with prior knowledge of the global environment [2], [3], [4], [5], [6], [7], or with estimated environment dynamics [12], [13], [14], or by pre-acting with visual imagination [15], [16], [17], [18]. These works either simply use self-assessment metrics to filter actions and re-choose the action with the next-highest affordance or handcrafted safer policy, or consume a large amount of data to build the environment or imagination module, which brings another problem of estimation bias. In contrast, our work directly uses the result representation from self-assessment during the training process, thus easily integrating self-assessment into our policy distribution.

Failure-aware Policy Learning. In this paper, we define the failure-aware policy to be aware of the previous action failures, and utilize the failed trials to predict more reliable actions. However, there are few studies under this definition. Hence, we review works that study policies predicting the success probability of the current action [13], [15], [16], [17], [18], [19], [20], or estimating the novelty and uncertainty of current observation [21], [22], [23], [24] which can be referred to as studies of failure-prediction policy. Other works like [14] predict the error of current action execution and propose an error-aware policy which takes as input the predicted future state error, and generates the corrected action. Similarly, [25] conducts an error detector by checking the reconstruction of the current state. However, most of these policies only measure the success probability of the current actions, without awareness of previous failed trials.

Robotic Exploration. Robotic exploration is a more general domain of our work, which can be regarded as exploration conditioned on the previous failures. Traditional works [26], [27], [28] design algorithms to explore states with less visiting times (*i.e* with larger entropy). Also, many recent works follow the similar idea to set bonus to states deemed to be interesting or novel [29], [30], [31], [32], [33], [34]. Other works like [1] propose to train a set of policies to overlap a group of contexts with a disagreement penalty. Another view is to decouple the exploration policy from the exploitation policy, thus eliminating the inductive bias from task reward and stabilizing the policy training [35].

III. Problem Formulation

In this work, we define self-assessable robotics tasks as those where a robot has some self-assessment rules to evaluate its performance before actually executing actions, and the evaluation results enable action re-choosing. Such self-assessment rules often serve as a safe module in real applications by predicting collision with simulation, which can provide a relatively accurate failure awareness. As a result, in this paper, we assume that self-assessment rules that correctly distinguish failure. Since there is no action execution, the observation stays invariant. Following the general framework in Fig. 1(a), in this section, we formulate and compare the process-of-elimination strategy and our method. Given the invariant observation o, a discrete action set $\mathcal{A}$, and a self-assessment module $\mathbf{SA}$, we can formulate the same part of the process-of-elimination strategy and our method as follows:

$$
\begin{aligned}
a_0 &= \pi_0(o)|_{a_0 \in \mathcal{A}} \\
f_t(a_t|o) := \mathbf{SA}\left(o, a_t\right)|_{t\geq 0} &= \begin{cases} 1, & \text{if successful} \\ 0, & \text{otherwise} \end{cases} \\
a_{t+1} &= \pi_{\mathrm{FA}}\left(f_t|\, o\right)|_{a_{t+1}\in\mathcal{A}}
\end{aligned}
\tag{1}
$$

where π_0 is the learned policy, a_t is the selected action at step t, f_t is a binary distribution defined by the $\mathbf{SA}$ results up to step t, and π_{FA} is a failure-aware policy conditioned on invariant observation o. Note that in this work, π_0 is assumed as a differentiable policy that contains an observation encoder and a decoder predicting the action affordances.

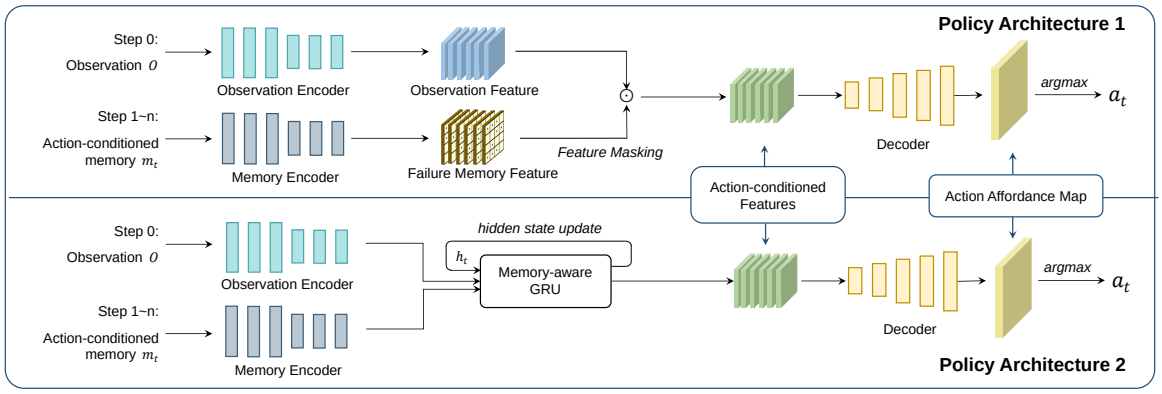


Fig. 2. Two architectures for the failure-aware policy π_{FA}. Note that the observation encoder and the decoder are components of the learned policy π_0, which accelerates the failure-aware policy training process.

Process-of-elimination. This is an intuitive way to design π_{FA} (Fig. 1(b)) which chooses the action with the next-highest affordance after figuring out that the selected action is unreliable, and can be formulated as follows:

$$a_{t+1} = \pi_{\mathrm{FA}}\left(f_t|\, o\right)|_{a_{t+1}\in\mathcal{A}} = \pi_0(o) * f_t(a_t|o)|_{a_{t+1}\in\mathcal{A}} \quad (2)$$

In this formulation, π_{FA} is a handcrafted sorting policy, which simply multiplies the learned policy distribution by f_t. In this way, the previous action failures do not influence the affordance distribution of the remaining untried actions.

Failure-aware Learning Policy. In this paper, we propose to integrate the self-assessment results into the failure-aware policy training, and use the result representation m_t from f_t as the variable state (Fig. 1(c)). Our framework can be represented as follows:

$$\begin{aligned} m_t &\sim f_t(a_t|o) \\ a_{t+1} &= \pi_{\mathrm{FA}}\left(m_t|\, o;\theta\right)|_{a_{t+1}\in\mathcal{A}} \end{aligned} \quad (3)$$

where π_{FA} is a learnable failure-aware policy, m_t is a representation of f_t with the same size as the action set $\mathcal{A}$, and θ represents the learnable parameters of neural network.

Self-assessment Representation. In our paper, m_t is used to represent the results of self-assessment. Concretely, it is a binary or normalized matrix of which each element represents the trial memory of the corresponding action, thus named action-conditioned memory. m_t is initialized at the beginning of every episode. For a binary m_t, it is initialized as an all-one matrix with the same size as the action set $\mathcal{A}$, while for a normalized representation, it is initialized as the normalized affordance map predicted by the learned policy π_0. If an action is assessed as failed during the trial process, then the corresponding element is set to zero, thus updating m_t during the whole episode.

IV. Methods

A. System Overview

Fig. 1(c) shows our pipeline. At the first step, the learned policy π_0 takes as input the observation o to get an initial action a_0. If a_0 fails according to the self-assessment module **SA**, a self-assessment result representation m_t will be constructed and fed into the failure-aware policy π_{FA} to re-choose another action a_t. Note that the network parameters of π_0 and π_{FA} are partially shared. Compared to the process-of-elimination strategy, we propose to investigate the dependency between the self-assessment results and the remaining untried actions by learning the failure-aware policy.

B. Failure-aware Policy Architecture

Considering that the memory of previous failure can be utilized either explicitly or implicitly, we propose two architectures for the failure-aware policy π_{FA}, which are shown in Fig. 2. Both of them contain an observation encoder and a decoder, which are the shared components of the learned policy π_0, which accelerates the training process.

Policy Architecture 1. The first proposed architecture is to encode the self-assessment result representation m_t into the same shape of the embedding feature generated from the observation encoder, and conduct an element-wise product to mask the observation feature by the failure memory feature embedding, which generates an action-conditioned feature. In this way, failure memory is explicitly considered into the feature embedding with a mask-like operation, thus affecting a shift on the action distribution.

$$\begin{aligned} e &= \mathbf{E}_{\mathrm{o}}(o) \odot \mathbf{E}_{\mathrm{m}}(m_t) \\ a_t &= \underset{a_t\in\mathcal{A}}{\operatorname{argmax}}\, \mathbf{D}(e) \end{aligned} \quad (4)$$

where $\mathbf{E}_{\mathrm{o}}$, $\mathbf{E}_{\mathrm{m}}$ and $\mathbf{D}$ symbolize the observation encoder, the memory encoder and the decoder respectively. a_t is selected from the action affordance map generated from $\mathbf{D}$. Note that the memory encoder can be simplified as a replica transform to the shape of the observation feature, or an identity transform in specific implementations.

Policy Architecture 2. The second architecture aims to bring the failure memory as a recurrent form across the episode using a memory-aware module. In this architecture, the feature embedding comes from the observation o at the first step, and from the updated m_t in the following steps. In this way, the failure memory is implicitly delivered across the decision process as a latent embedding.

$$e = \begin{cases} \mathbf{E}_{\mathrm{o}}(o) & t = 0 \\ \mathbf{E}_{\mathrm{m}}(m_t) & t > 0 \end{cases} \quad (5)$$

where $\mathbf{E}_{\mathrm{o}}$ and $\mathbf{E}_{\mathrm{m}}$ are of the same definitions in Eq. 4. With the feature embedding, the memory-aware module obtains

awareness of the observation at the first step and implicitly represents it by the hidden vector, then produces new action distributions with recurrent memory in the following steps.

$$a_t = \underset{a_t \in \mathcal{A}}{\operatorname{argmax}} \mathbf{D}\left(\mathbf{GRU}(e)\right) \tag{6}$$

where $\mathbf{D}$ is of the same definition in Eq. 4, and $\mathbf{GRU}$ corresponds to the memory-aware module.

C. Policy Learning

We implement behavior cloning to train the learned policy π_0. Note that there is no sequential decision making for this stage. For each step, the policy generates an action under the observation and gets feedback from self-assessment. Also, since parameters of π_0 and π_{FA} are partially shared, it can be seen as pre-training for the failure-aware policy π_{FA}.

To train the failure-aware policy, we apply value-based RL algorithms. For each episode, the policy is provided with the observation at the first step and chooses an action. If the feedback from the self-assessment module $\mathbf{SA}$ is positive, then the episode ends. Otherwise, the failure-aware policy π_{FA} will take as input the memory representation m_t and re-choose an action until receiving positive feedback from $\mathbf{SA}$. During the training process, network parameters shared with π_0 will be fixed.

V. Experimental Results

In this section, we will conduct experiments in three self-assessable robotics tasks to: 1) evaluate the effectiveness and advantages of our failure-aware policy compared with other methods; 2) show the different performances of the two failure-aware policy architectures; 3) investigate what kind of policy is optimal for the sequential decision making problem under invariant observation.

A. Experimental Setup

We consider three typical self-assessable robotics tasks (Fig. 3) for evaluation. The first task is sequential image classification on ImageNet [36] motivated by [1]. In this task, the robot observes an image at the beginning of an episode, and identifies a label for this image. After choosing a label, the self-assessment module will indicate whether the choice is correct or not. The second task is object reorientation, where a robot is supposed to choose a reorientation object pose to achieve a feasible pick-reorient-place process (*i.e.* successful path planning of the whole manipulation) with path planning cost as less as possible [13]. For this task, the policy is trained in SAPIEN [37] with a UR5 arm, tested with unseen samples, and evaluated in real world. Self-assessment is conducted by path planning algorithms, which guarantee the execution success. And the last is localization on synthetic dataset [38] and real-world dataset UPO [39] and Bicocca [40], [41], which predicts the position of the robot given a global map and an observed scan, and gets the assessment of the localization accuracy. In this task, if the predicted position is at the $k \times k$ neighborhood of the ground truth position, then the action is regarded as successful. Note that the robot will re-choose action after getting the self-assessment results until

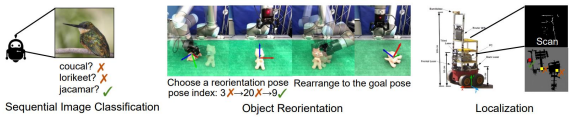


Fig. 3. Three tasks for evaluation.

TABLE I
Testing Performance of Sequential Image Classification

Method	tsr/%	tns
RE	0.54	2.93
LPRE	70.06	**1.01**
SP	**89.05**	1.39
FMP-1	**89.05**	1.39
FMP-2	**89.07**	1.39

evaluated as successful or up to the limited trial times. In real applications, we can use registration algorithms as the self-assessment module which measures localization accuracy. Details of the self-assessment module of these tasks, the learned policy, and implementations of our two architectures as well as the training and testing settings can be found in Appendix.

B. Metrics and Baselines

In this work, we aim to achieve a reliable action as soon as possible, since online self-assessment costs time and energy. Thus, we limit the sequential trial number to $t = 5$ times, and measure the algorithms with the following metrics.

- **Task Success Rate (tsr)**: Average task success rate across all testing samples. If the policy passes the self-assessment within 5 trials, then the corresponding testing sample is regarded as successful.
- **Trial Number to Success (tns)**: Average trial number to get a success feedback from self-assessment of all success samples.

All the tasks are measured with **tsr** and **tns**, which demonstrate the effectiveness and efficiency of action re-choosing based on previous failed trials. Also, an additional metric is tested for the object reorientation task:

- **Planning Cost (pc)**: Average path planning cost across all testing samples. It is a unique metric for object reorientation task, where robot is supposed to choose a reorientation pose with path planning cost as less as possible.

We compare the performance of our system to the following baseline approaches:

Random Exploration (RE). A policy which selects actions uniformly at random from the candidate action set $\mathcal{A}$.

Learned Policy with Random Exploration (LPRE). A policy that uses the learned policy π_0 for the first step, and if failed, then use the **RE** policy among the remaining actions.

Sorting Policy (SP). A policy that uses the process-of-elimination strategy.

Also, we name our policies of two architecture as **Failure-aware Feature Masking Policy (FMP-1)** and **Failure-aware Recurrent Memory Policy (FMP-2)** respectively.

C. Results

Comparisons to Baselines. First, we compare our method with baselines in three tasks. For sequential image classifica-

TABLE II

TESTING PERFORMANCE OF OBJECT REORIENTATION

Method	tsr/%	100/pc	tns
RE	77.50	3.07	2.18
LPRE	79.33	3.48	1.90
SP	81.64	3.25	2.05
FMP-1	86.96	3.91	1.77
FMP-2	**89.37**	**4.55**	**1.61**

TABLE III

TESTING PERFORMANCE OF LOCALIZATION IN SYNTHETIC ENVIRONMENTS.

Method	tsr/%	tns
LPRE	83.95	**1.01**
SP	84.23	**1.01**
FMP-1	**94.53**	1.47
FMP-2	85.02	1.04

tion, we evaluate each method with the validation dataset of ImageNet [36]. Note that in this task we regard the output of π_0 as the observation feature embedding (more details can be found in Appendix). That is, **SP** and **FMP-1** have the same settings for this task, thus reporting the same performances. We can see from Table I that except for **RE**, **LPRE** shows the worst performance, while other three methods demonstrate similar performances. Referring to the analysis in [1], which proves that **SP** is an optimal policy for the sequential image classification task, our experimental results further figure out that **FMP-1** and **FMP-2** can also achieve optimal performance for this task.

For the object reorientation experiments, we present an additional metric "pc" in the reciprocal form, since the path planning cost will be infinite if the planning fails. Each method is evaluated with 207 unseen samples. Results in Table II show that **FMP-2** outperforms other methods across all metrics, followed by **FMP-1**, which demonstrates that integrating the previous failure memory into policy training endows better policy tune-up during online testing and better generalization performance. Also, **FMP-2** reports better performance than **FMP-1** across all metrics. This might be due to the fact that, in this task, there exist some candidate reorientation poses which are similar to each other, thus leading to a similar point cloud feature. However, similar poses do not mean similar task assessment results. For example, flipping an object will fail due to the collision between the gripper and the table. But if the object is with a similar pose which leaves small space for the gripper, such manipulation might succeed. For **FMP-1**, it utilizes previous failures by conducting feature masking, which might hinder some possible successful actions. Moreover, randomly choosing actions among the remaining ones (**LPRE**), or applying memory as a mask (**SP**) neglects the dependency between the failure trials and the remaining actions, which shows lower performance compared to our policies. Besides, we can find that the performance **RE** is not too bad due to the small action set of this task.

As for the localization task, the testing localization samples include three testing sequences in synthetic environments [38] and two testing sequences in real environments. Evaluation results in synthetic environments are shown in Table III

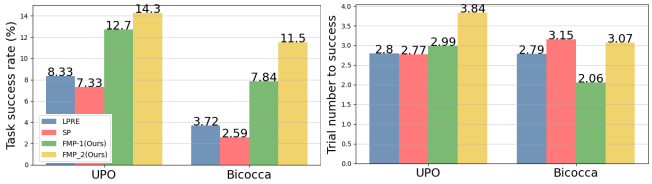


Fig. 4. Testing Performance of Localization in Real Environments.

($k = 15$, more detailed results and ablation studies on k can be found in Appendix). It is obvious that **FMP-1** achieves the best performance in the episode success rate with an average trial number less than 2. This large performance margin might come from the feature masking process, which hinders the similar feature of failed actions and encourages the policy to jump from the previous choices. **FMP-2** also shows better task success rate compared to **SP**. Note that the trial numbers to success of **LPRE** and **SP** are both close to 1, which indicates that these two policies cannot conduct effective adjustment by the self-assessment results. Fig. 4 demonstrates the testing results in real environments, which shows that our policies have better task success rate than **SP** and **LPRE** in all real environments. Overall, **FMP-2** shows better episode success rate, while **FMP-1** costs less trials to locate the position. This might be because the complex geometry of real-world maps calls for exploration in a small scope, which is the advantage of **FMP-2**, while **FMP-1** might hinder these similar positions if one of them fails. However, by jumping from the previous failure zone, **FMP-1** is able to achieve success in less trials. Note that we use $k = 15$ because the complex geometry of real-world maps leads to multi-modal predictions, and we choose the action with maximum affordance in our experiments.

Case Studies. Fig. 5 presents some testing cases in the localization task of three policies. Since **SP** does not change its original distribution, all its decisions depend on the distribution generated from π_0. Instead, **FMP-1** concerns more on the feature correlation. When aware of a failed action, it can hinder the similar feature, thus jumping out of the previous failure zone. Also, by leveraging the recurrent implicit memory, **FMP-2** is also capable of adjusting the decisions, but shows a more conservative exploration process. Also, we can see the normalized probability changes of all feasible reorientation poses in an object reorientation case in Fig. 6. In this task, the feasible poses are not unique, and the similar poses (with near pose indexes) do not mean similar feasibility. In this task, **FMP-2** performs better because it tends to explore the near pose first to confirm its feasibility. More case studies are shown in Appendix.

D. Discussion

Considering the architecture designs of our models and all the experimental results, we further analyze the advantages of our method, and how to choose an optimal policy in a specific tasks:

What are the advantages of two policy architectures? The same advantage of these two policy architectures is learnable. By integrating the self-assessment results into the policy training, **FMP-1** and **FMP-2** acquire the awareness of

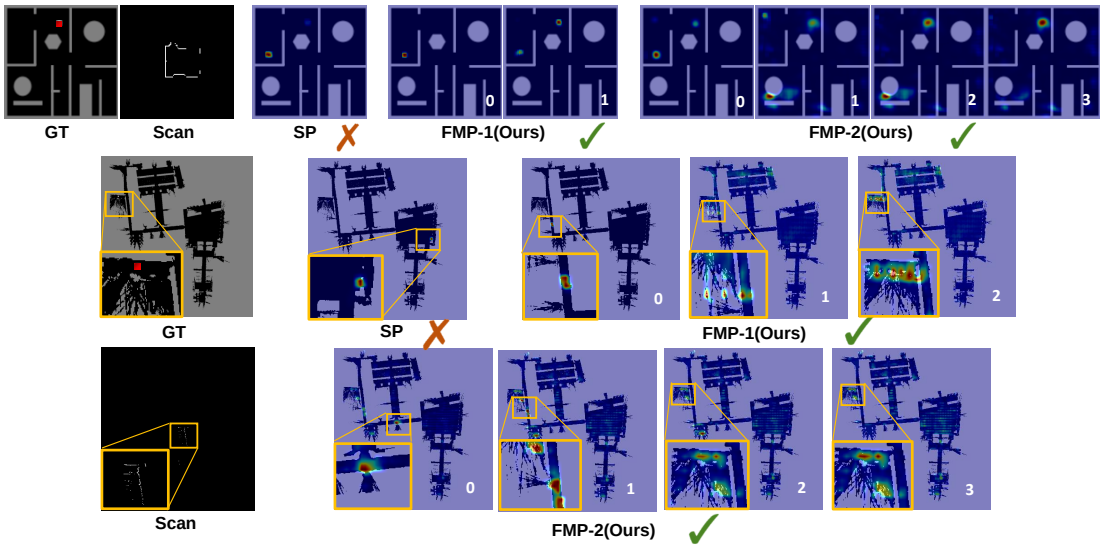


Fig. 5. Testing cases in localization task of three policies. The top row is a case in a synthetic environment. The left two columns show the global map with the ground truth position labeled as a red point, and the scan observation. Other columns show the prediction process and the distributions of three policies. The remaining row shows a case in the UPO dataset, where the keys are zoomed in with yellow boxes. The distribution is reflected by the color, where the value comes larger as the color comes closer to red. ✓means that the policy successfully find the right position, while ✗means a failure.

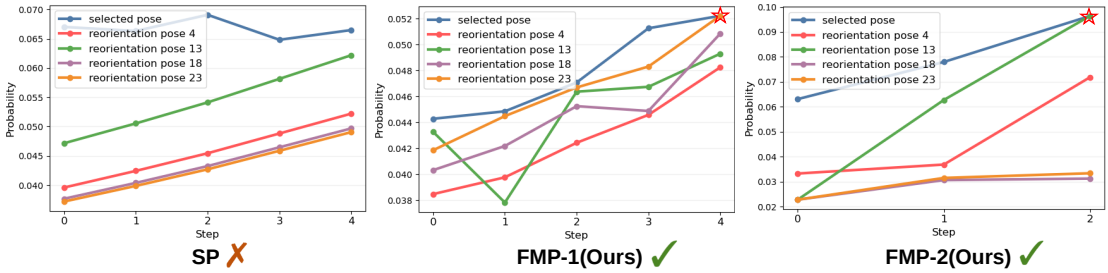


Fig. 6. A Testing case in reorientation task of three policies. The setting of this case is: magic clean, initial pose: (-0.16, 0.16, 0.03, 1.59, 0.01, -3.14), target pose: (-0.15, 0.09, 0.13, 0.0, 0.0, 0.0), feasible reorientation poses indexes: (4, 13, 18, 23). We plot the normalized probability distributions of feasible poses the selected pose at all trial steps, and the stars label successful trials. ✓means that the policy successfully finds the right pose, while ✗means a failure. The decision sequences of three policies are **SP**: 12→7→2→17→8, **FMP-1**: 22→15→3→2→23, **FMP-2**: 24→17→13.

the previous failures, and have the capability of trimming the policy distribution according to these failures. However, these two architectures show different properties. **FMP-1** concerns more on the feature correlation, and tends to hinder the actions with similar features to that of the previously failed choices. Thus, **FMP-1** can achieve better performance in tasks where similar features (*e.g.* pose, geometry, visual attribute, and etc) lead to similar self-assessment results. Instead, **FMP-2** pays more attention to the recurrent memory, which encodes the observation feature and the previous trials. Consequently, **FMP-2** conducts a more conservative exploration process than **FMP-1**, and demonstrates better performances in tasks requiring an adjustment in a small scope.

What kind of policies is optimal? For a task where similar features lead to similar self-assessment results, **FMP-1** can help jump out of local minimum, while for a task which needs to adjust in a small scope of the initial action, **FMP-2** shows better exploration strategy. Also, analyzing the results of the three tasks, we figure out that, when the actions are correlated, our method outperforms the process-of-elimination strategy. That is, *when the actions are correlated, the equality is broken between the action with the next-highest affordance and the action with the highest affordance conditioned on previous failures*. For sequential image classification, there is little correlation among different class choices. And the training data is adequate for fitting the distributions of all classes. Therefore, a simple process-of-elimination strategy can achieve optimal performance. Instead, as the action correlation increases, integrating action correlation into the policy learning endows better performance. Hence, we can achieve better performances in the localization task, and show the biggest advantage in the object reorientation task, whose actions have the highest correlation.

E. Conclusion and Limitation

In this paper, we propose to integrate the self-assessment results to learn a failure-aware policy, and propose two policy architectures. Experiments in three self-assessable robotics tasks demonstrate that our method outperforms other methods with higher task success rate with less trials. Moreover, we find that the action correlation has a large impact on the effect of our algorithm. The main limitation of our method lies in the assumption of the discrete action set. This limitation comes from the construction of the representation m_t. In our paper, it is represented as a finite matrix, of which each element corresponds an action. In future work, more general representations of m_t for continuous actions can be studied. Also, in this paper, we assume that the self-

assessment module can accurately predict failure, and π_0 is a learned differentiable policy with a feature bottleneck layer. Further works can involve the assessment uncertainty in more real-world applications, and extend to more general π_0.

REFERENCES

[1] D. Ghosh, J. Rahme, A. Kumar, A. Zhang, R. P. Adams, and S. Levine, "Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[2] D. Isele, A. Nakhaei, and K. Fujimura, "Safe reinforcement learning on autonomous vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–6.

[3] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn, "Learning to be safe: Deep rl with a safety critic," *arXiv preprint arXiv:2010.14603*, 2020.

[4] H. Krasowski, X. Wang, and M. Althoff, "Safe reinforcement learning for autonomous lane changing using set-based prediction," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–7.

[5] K. Mokhtari and A. R. Wagner, "Safe deep q-network for autonomous vehicles at unsignalized intersection," *arXiv preprint arXiv:2106.04561*, 2021.

[6] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1774–1783.

[7] D. Chen, Z. Li, Y. Wang, L. Jiang, and Y. Wang, "Deep multi-agent reinforcement learning for highway on-ramp merging in mixed traffic," *arXiv preprint arXiv:2105.05701*, 2021.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *EMNLP*, 2014.

[10] G. J. Burghouts, A. Huizing, and M. A. Neerincx, "Robotic self-assessment of competence," *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2020.

[11] A. Norton, H. Admoni, J. Crandall, T. Fitzgerald, A. Gautam, M. Goodrich, A. Saretsky, M. Scheutz, R. Simmons, A. Steinfeld *et al.*, "Metrics for robot proficiency self-assessment and communication of proficiency in human-robot teams," *ACM Transactions on Human-Robot Interaction*, 2022.

[12] P. Deptula, H.-Y. Chen, R. A. Licitra, J. A. Rosenfeld, and W. E. Dixon, "Approximate optimal motion planning to avoid unknown moving avoidance regions," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 414–430, 2019.

[13] K. Xu, H. Yu, R. Huang, D. Guo, Y. Wang, and R. Xiong, "Efficient object manipulation to an arbitrary goal pose: Learning-based anytime prioritized planning," *2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[14] V. Kumar, S. Ha, and C. K. Liu, "Error-aware policy learning: Zero-shot generalization in partially observable dynamic environments," *Robotics: Science and Systems (RSS)*, 2021.

[15] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2786–2793.

[16] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *arXiv preprint arXiv:1812.00568*, 2018.

[17] A. Wang, T. Kurutach, K. Liu, P. Abbeel, and A. Tamar, "Learning robotic manipulation through visual planning and acting," in *Robotics: science and systems*, 2019.

[18] N. Di Palo and E. Johns, "Safari: Safe and active robot imitation learning with imagination," *arXiv preprint arXiv:2011.09586*, 2020.

[19] K. Xu, H. Yu, Q. Lai, Y. Wang, and R. Xiong, "Efficient learning of goal-oriented push-grasping synergy in clutter," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6337–6344, 2021.

[20] Z. Xu, Z. He, J. Wu, and S. Song, "Learning 3d dynamic scene representations for robot manipulation," in *Conference on Robot Learning*. PMLR, 2021, pp. 126–142.

[21] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," 2017.

[22] L. Wellhausen, R. Ranftl, and M. Hutter, "Safe robot navigation via multi-modal anomaly detection," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1326–1333, 2020.

[23] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[24] B. Lütjens, M. Everett, and J. P. How, "Safe reinforcement learning with model uncertainty estimates," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8662–8668.

[25] J. Wong, A. Tung, A. Kurenkov, A. Mandlekar, L. Fei-Fei, S. Savarese, and R. Martín-Martín, "Error-aware imitation learning from teleoperation data for mobile manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 1367–1378.

[26] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos, "Active policy learning for robot planning and exploration under uncertainty." in *Robotics: Science and systems*, vol. 3, 2007, pp. 321–328.

[27] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," *Advances in neural information processing systems*, vol. 29, 2016.

[30] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, 2016.

[31] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International conference on machine learning*. PMLR, 2017, pp. 2778–2787.

[32] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[33] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[34] M. C. Machado, M. G. Bellemare, and M. Bowling, "Count-based exploration with the successor representation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5125–5133.

[35] W. F. Whitney, M. Bloesch, J. T. Springenberg, A. Abdolmaleki, K. Cho, and M. Riedmiller, "Decoupled exploration and exploitation policies for sample-efficient reinforcement learning," *arXiv preprint arXiv:2101.09458*, 2021.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[37] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang *et al.*, "Sapien: A simulated part-based interactive environment," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 097–11 107.

[38] R. Chen, H. Yin, Y. Jiao, G. Dissanayake, Y. Wang, and R. Xiong, "Deep samplable observation model for global localization and kidnapping," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2296–2303, 2021.

[39] R. Ramón-Vigo, J. Pérez, F. Caballero, and L. Merino, "Navigating among people in crowded environment: Datasets for localization and human robot interaction," in *Proceedings of the Workshop on Robots in Clutter: Perception and Interaction in Clutter, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Citeseer, 2014.

[40] A. Bonarini, W. Burgard, G. A. E. Fontana, M. Matteucci, D. Sorrenti, and J. Tardos, "Rawseeds: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets," in *Workshop on Benchmarks in Robotics Research at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, 2006, pp. 1–5.

[41] S. Ceriani, G. Fontana, A. Giusti, D. Marzorati, M. Matteucci, D. Migliore, D. Rizzi, D. G. Sorrenti, and P. Taddei, "Rawseeds ground truth collection systems for indoor self-localization and mapping," *Autonomous Robots*, vol. 27, no. 4, pp. 353–371, 2009.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[43] R. Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

APPENDIX I
EXPERIMENT IMPLEMENTATION DETAILS

In this section, we will introduce three typical robotics tasks (shown in Fig. 3) that we use in experiment evaluation, and demonstrate how to apply our algorithm in these specific tasks.

A. Sequential Image Classification

Task Definition. Our first task is sequential image classification, which is motivated by [1] and the setting is also similar. In this task, the robot observes an image from the dataset at the beginning of an episode and identifies a label for this image. After choosing a label, the robot will receive feedback from the self-assessment module about whether the choice is correct. In real applications, the assessment process can be conducted by human-robot interaction. If incorrect, the robot is supposed to re-choose a label until evaluated as correct by the self-assessment module.

Policy Architecture Implementation. In this task, we use the pre-trained model of ResNet18 [42] on ImageNet [36] from [43] as the learned policy π_0. Self-assessment representation m_t is a normalized vector with the same size as the output of π_0, which is initialized as the softmax output of π_0. And for **FMP-1**, we use π_0 (*i.e.* ResNet18) as the observation encoder, whose output can be regarded as the observation embedding feature. And the memory encoder and the decoder are identity transforms. Thus, such an implementation of **FMP-1** is equivalent to that of process-of-elimination [1] (*i.e.* first choosing the action with the highest affordance, if incorrect, then the second, and so forth). For **FMP-2**, we use π_0 (*i.e.* ResNet18) as the observation encoder and a GRU [9] as the memory-aware module. Memory encoder and decoder are both identity transforms.

Training and Testing Details. We apply DQN [44] to train π_{FA} with shared parameters of π_0 fixed. For each episode, the robot is provided with an image from the ImageNet training set, and gives a sequence of guesses for the image label among the total 1000 labels. At training time, if the selected label is correct, the robot gets a reward of $r = 1$, and the episode ends. Otherwise, the robot gets a reward of $r = 0$, and the episode continues to the next time step. We limit the trial times up to $t = 5$. We train the network with SGD optimizer with L1 loss, using learning rate of 10^{-4}, momentum of 0.9 and weight decay 2^{-5}, and the future discount γ is constant at 0.2. At testing time, we evaluate the trained policy with the validation set of ImageNet (50k images) with the same episode setting as training.

B. Object Reorientation

Task Definition. Object reorientation is a manipulation task where a robot is supposed to choose a reorientation object pose to achieve a feasible or even optimal pick-reorient-place process [13]. In this task, the robot is provided with an assigned object, with its mesh model, initial pose and target pose, and is supposed to choose a reorientation pose as a transition since the one-step pick-and-place might be failed due to collision between the robot and the environment. The policy in [13] first generates a finite set of reorientation pose candidates and predicts an affordance map of these poses, and the one with the highest affordance will be conducted. Also, pre-acting in a simulator with the planning algorithm RRT predicts the path planning cost of the whole pick-reorient-place process after choosing a reorientation pose, which serves as the self-assessment module. In this task, we set the reorientation pose candidate number $n = 25$.

Policy Architecture Implementation. We use the policy in [13] as the learned policy π_0. Self-assessment representation m_t is a binary vector with the same size as the candidate pose set. For **FMP-1**, the feature extraction module (NE) of π_0 followed by a self-attention layer is used as the observation encoder, which outputs n embedding concated features corresponding to n pose candidates. The decoder is the evaluation module (PCEN) of π_0. And the memory encoder is a replica transform, which converts the shape of m_t to the same as the observation feature. For **FMP-2**, the observation encoder is that of **FMP-1** followed by additional convolution blocks, the decoder and the memory encoder are identity transforms, and the memory-aware module is a GRU [9]. The network architectures are presented in Fig. 14.

Training and Testing Details. π_0 is pre-trained via behavior cloning with a dataset containing 3048 samples labeled with the path planning costs. To build this dataset, we collect each sample in SAPIEN [37] with a UR5 arm by randomly choosing an object model from the model set consisting of 21 3D object models, with an initial pose and a target pose randomly sampled from the stable place poses. Then the robot traversely executes the planned pick-reorient-place trajectories of all the reorientation pose candidates, which generates the planning costs of all the "actions". The planning algorithm is RRT, and the planning cost is numerically dependent on the trajectory's existence and length. To train π_{FA}, DQN [44] is applied with parameters of NE fixed. For each episode, a model is randomly sampled from the model set, with an initial pose and a goal pose randomly sampled from the stable place poses. Then the policy chooses a reorientation object pose and executes the planned trajectories. If planning fails, another pose will be selected until the planning succeeds or up to $t = 5$ trials. The reward design is the same as that in [13]. We train the network with Adam optimizer with Huber loss, using learning rate 10^{-4}, weight decay 2^{-5}, betas $(0.9.0.99)$, and the future discount γ is constant at 0.2. At testing time, we evaluate the trained policy with 207 unseen samples with the same episode setting. The real-world environment contains a UR5 arm, and an example sequence is shown in Fig. 7.

C. Localization

Task Definition. We also evaluate our method with a typical mobile robotics task: localization. The task setting is the same as in [38]. In this task, the robot predicts its position given a global map and an observed scan. And the self-assessment module identifies the localization accuracy. In real applications, we can use some registration algorithms as self-assessment module which measures the localization

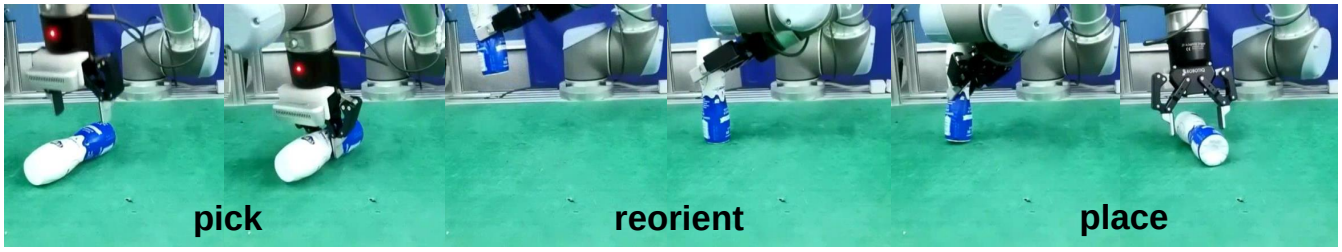
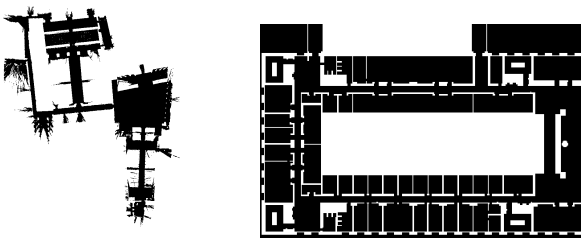


Fig. 7. An example pick-reorient-place sequence in real-world environment.



(a) UPO (b) Bicocca

Fig. 8. Real-world global maps for the localization task.

accuracy. In our experiments, though, we use datasets to provide ground truth for convenience. In this task, if the predicted position is at the $k \times k$ neighborhood of the ground truth position, the action is regarded as successful. Otherwise, the robot will re-choose the position to reach the localization accuracy. Synthetic datasets and real-world datasets (UPO [39] and Bicocca [40][41] shown in Fig. 8) are used to train and test as the same way in [38]. Following [38], Bicocca is split into two datasets. The original images are resized into $H \times W$ ($H = W = 128$) before feeding into the network.

Policy Architecture Implementation. We use the pre-trained models in [38] as the learned policy π_0. Self-assessment representation m_t is a binary mask with shape $H \times W$, which is the shape of the global map. m_t is updated with neighborhood $k \times k$. For **FMP-1**, we use the encoder and part of the decoder of π_0 as the observation encoder and the remaining decoder of π_0 as the decoder. And memory encoder is an identity transform. For **FMP-2**, the observation encoder and the decoder are the same as π_0, with a separate memory encoder with the same network architecture as the observation encoder, and a GRU [9] as the memory-aware module. The network architectures are presented in Fig. 15.

Training and Testing Details. We apply DQN [44] to train π_{FA} with parameters of observation encoder fixed. For each episode, a global map and an observed scan are fed into the policy, and an initial affordance map with the same size as the global map is predicted at the first step. If failed, m_t will be updated and re-decision will be conducted by π_{FA} in the following steps. Successfully reaching the localization accuracy or coming up to the limited trial times ($t = 5$) ends the episode. At training time, if the selected position is evaluated as successful, the robot gets a reward of $r = 1$, and the episode ends. Otherwise, the robot gets a reward of $r = 0$, and the episode continues to the next step. We train the network with Adam optimizer with smooth L1 loss, using learning rate of 10^{-3}, weight decay 2^{-6}, and the future discount γ is constant at 0.2. At testing time, we evaluate the trained policy with the same episode setting as training. Three sequences of synthetic data in two unseen synthetic maps, and three sequences of real-world data in three real-world maps with unseen observation are used as the validation set.

TABLE IV
TESTING PERFORMANCE OF LOCALIZATION IN SYNTHETIC ENVIRONMENTS.

Method	k	tsr/%				tns/%			
		S-1	S-2	S-3	avg	S-1	S-2	S-3	avg
LPRE	15	91.54	74.16	86.15	83.95	1.03	**1.00**	**1.00**	**1.01**
SP		90.77	75.00	86.92	84.23	**1.00**	1.01	1.01	**1.01**
FMP-1		**98.46**	**86.67**	**98.46**	**94.53**	1.22	1.75	1.45	1.47
FMP-2		90.77	75.83	88.46	85.02	1.01	1.09	1.03	1.04
LPRE	9	89.23	74.16	83.85	82.41	**1.00**	**1.00**	**1.00**	**1.00**
SP		90.77	75.00	85.38	83.72	1.03	1.01	1.05	1.03
FMP-1		**91.54**	**84.59**	**93.85**	**89.99**	1.08	1.65	1.13	1.29
FMP-2		87.69	79.17	83.85	83.57	1.01	1.01	1.02	1.01
LPRE	5	71.54	52.50	72.31	65.45	**1.00**	**1.00**	**1.00**	**1.00**
SP		84.61	69.17	81.54	78.44	1.31	1.37	1.14	1.27
FMP-1		**86.15**	**76.67**	**91.54**	**84.79**	1.21	1.46	1.44	1.37
FMP-2		66.15	59.17	66.15	63.82	2.38	1.17	1.01	1.08

[2] * S-1, S-2, S-3 represent three synthetic testing sequences.

APPENDIX II
MORE EXPERIMENTAL RESULTS

A. Ablation Studies

Localization Neighborhood Size. We conduct an ablation study on the neighborhood size in the localization task (shown in Table. III, where $k = 5, 9, 15$). We can see that **FMP-1** shows the best performance across all neighborhood sizes. And the advantage becomes greater as localization accuracy increases (*i.e.* k decreases). Instead, **FMP-2** shows better or comparable performances compared to **SP** when $k = 9, 15$, but presents worst performance when $k = 5$. This might be because **FMP-1** concerns more with the recurrent memory across the sequential decision process, thus tending to choose the position near the previously chosen ones. Under high localization accuracy requirements, such a strategy might be stuck in the local minimum.

Another Architecture Implementation. In this paper, we propose two policy architectures and implement them in three specific tasks. Actually, there are many implementations of the two architectures. In this part, we provide another

TABLE V

TESTING PERFORMANCE OF LOCALIZATION IN SYNTHETIC ENVIRONMENTS WITH DIFFERENT IMPLEMENTATIONS OF POLICY ARCHITECTURE 1.

Method	tsr/%				tns/%			
	S-1	S-2	S-3	avg	S-1	S-2	S-3	avg
FMP-1	**98.46**	86.67	**98.46**	94.53	1.22	1.75	1.45	1.47
FMP-1.5	96.92	**95.83**	94.62	**95.79**	**1.13**	**1.60**	**1.21**	**1.31**

[2] * S-1, S-2, S-3 represent three synthetic testing sequences.

TABLE VI

TESTING PERFORMANCE OF LOCALIZATION IN REAL-WORLD ENVIRONMENTS WITH DIFFERENT IMPLEMENTATIONS OF POLICY ARCHITECTURE 1.

Method	tsr/%		tns/%	
	U	B	U	B
FMP-1	12.67	**7.84**	**2.99**	**2.06**
FMP-1.5	**31.00**	6.89	**2.99**	3.64

[2] * U, B represent UPO, Bicocca respectively.

implementation of Policy Architecture 1 in the localization task, which is named **FMP-1.5**. In this implementation, the memory encoder is of the same architecture as the observation encoder with other designs same as **FMP-1**. Compared results in three synthetic environments and three real-world environments are shown in Table V and Table VI. In synthetic environments, **FMP-1.5** shows comparable average episode success rates to **FMP-1** with less average trials. In real-world environments, **FMP-1** shows better performance in Bicocca maps. However, **FMP-1.5** achieves more than twice that of **FMP-1** in episode success rate with about the same trial times. Overall, **FMP-1.5** shows better performance in synthetic maps and UPO map, which demonstrates the advantage of using an embedding representation of m_t. But in Bicocca maps with relatively more symmetric structures, which lead to multi-modal predictions, directly leveraging m_t is more effective. This might be due to the fact that a proper embedding representation of m_t might cost more training samples.

B. Case Visualization

Fig. 9 and Fig. 10 present more testing cases of the localization task in synthetic environments and real-world environments, which further demonstrate the advantage of our failure-aware policies and the different properties of the two policy architectures. Also, Fig. 10 is a failure case of **FMP-2**, which explores with a conservative way in this case. Also, Fig. 11 shows the distribution changes at all trial steps of three policies.

C. Detailed Results

We provide detailed results of Table III in Table IV, which presents the results of all synthetic testing sequences. Also, a clear figure of Fig. 5 is shown as Fig. 13. And Fig. 12 shows the predicted probability distribution changes of all candidate reorientation poses of Fig. 6.

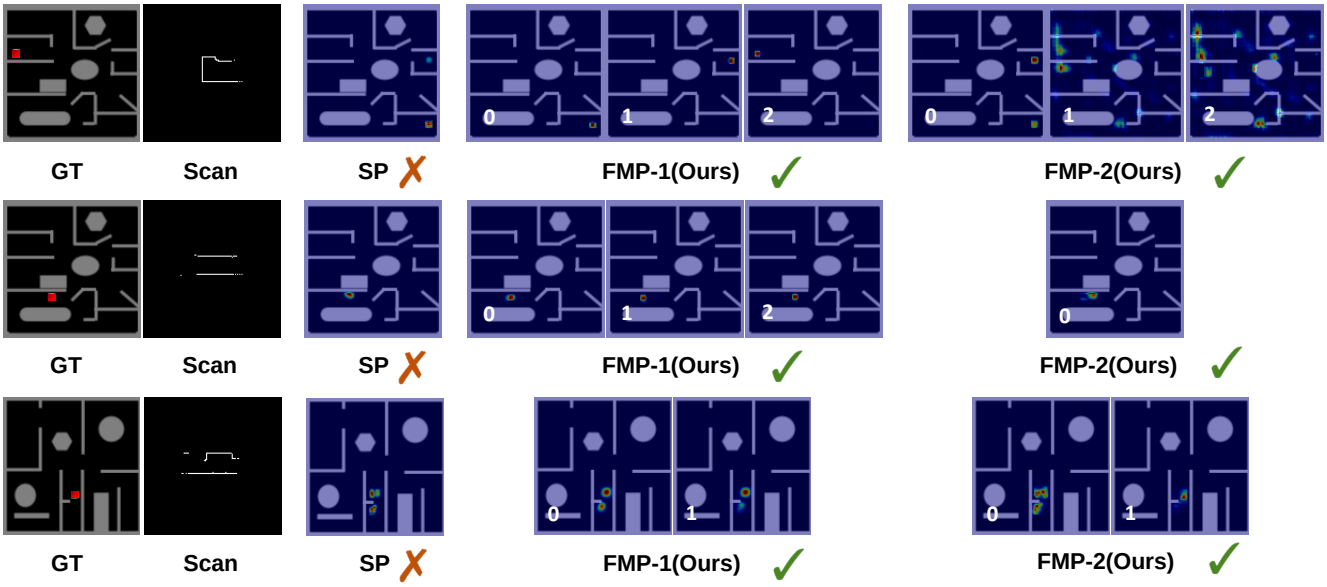


Fig. 9. Testing cases of localization task in synthetic environments of three policies. The left two columns show the global map with the ground truth position labeled as a red point, and the scan observation. Other columns show the prediction process and the distributions of three policies. The distribution is reflected by the color, where the value comes larger as the color comes closer to red. ✓means that the policy successfully finds the right position, while ✗means a failure.

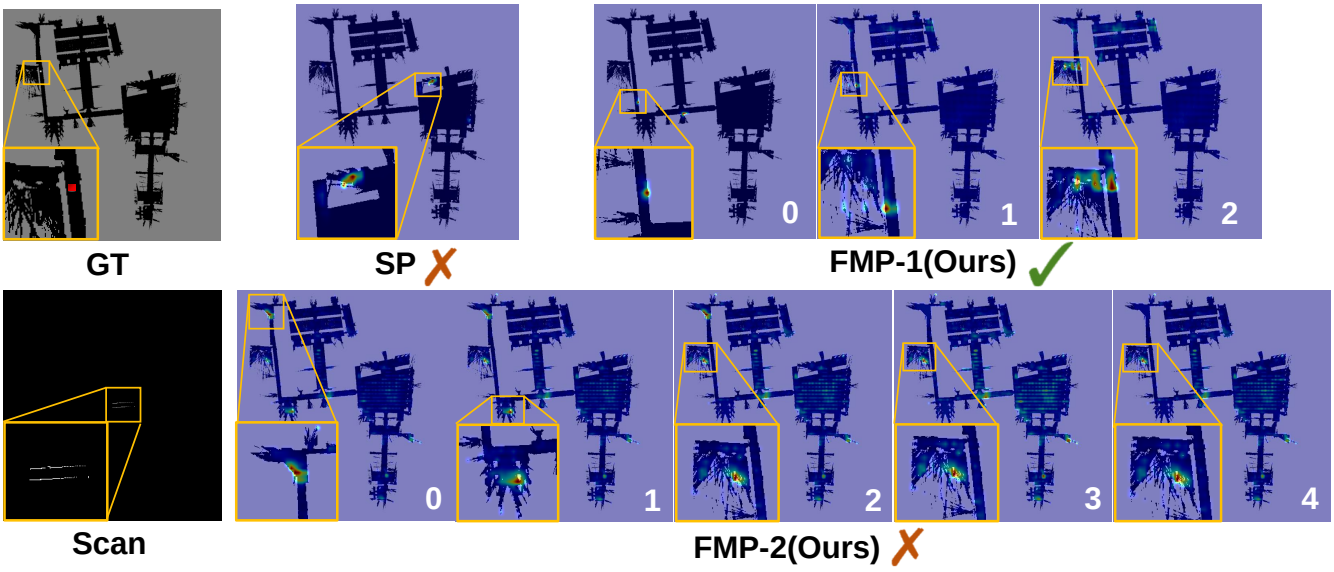
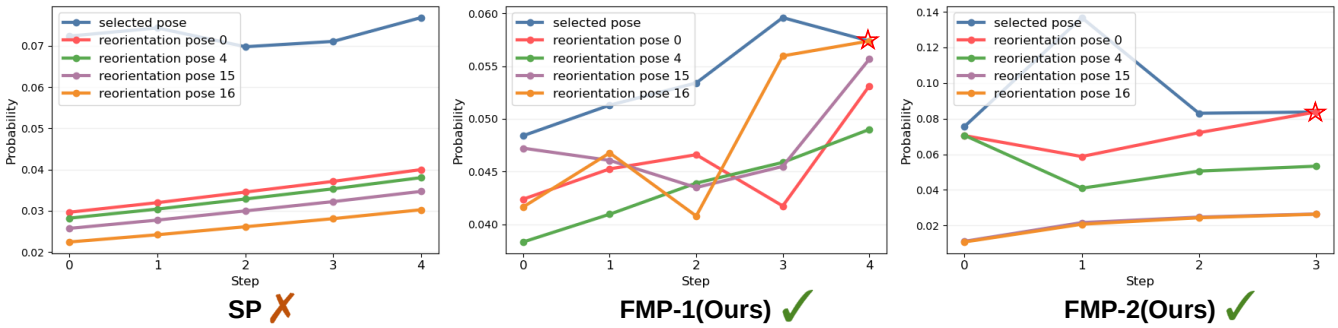
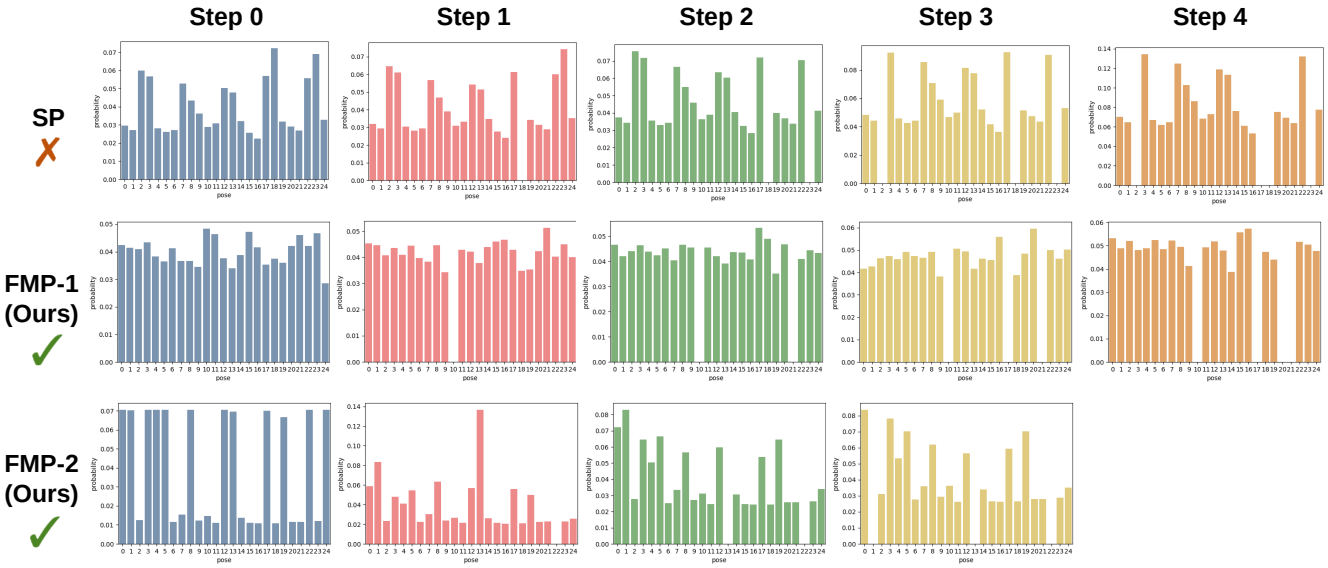


Fig. 10. A testing case of localization task in real-world environments of three policies. The left column shows the global map with the ground truth position labeled as a red point, and the scan observation. Other columns show the prediction process and the distributions of three policies. The distribution is reflected by the color, where the value comes larger as the color comes closer to red. ✓means that the policy successfully finds the right position, while ✗means a failure.



(a) Normalized probability distributions of all feasible poses the selected pose at all trial steps, and the stars label successful trials.



(b) Normalized probability distribution changes of the remaining actions at all trial steps of three policies.

Fig. 11. A testing case in reorientation task of three policies. Each case has several feasible poses. ✓means that the policy successfully finds the right position, while ✗means a failure. The settings of this case are: conditioner, initial pose: (0.20, -0.29, 0.02, 1.56, -1.51e-05, -3.14), target pose: (-0.05, -0.09, 0.03, 1.04, 1.46, 2.61), gt poses: (0, 4, 15, 16). In this case, the decision sequence of three policies are **SP**: 18→23→2→17→3, **FMP-1**: 10→21→17→20→16, **FMP-2**: 22→13→1→0.

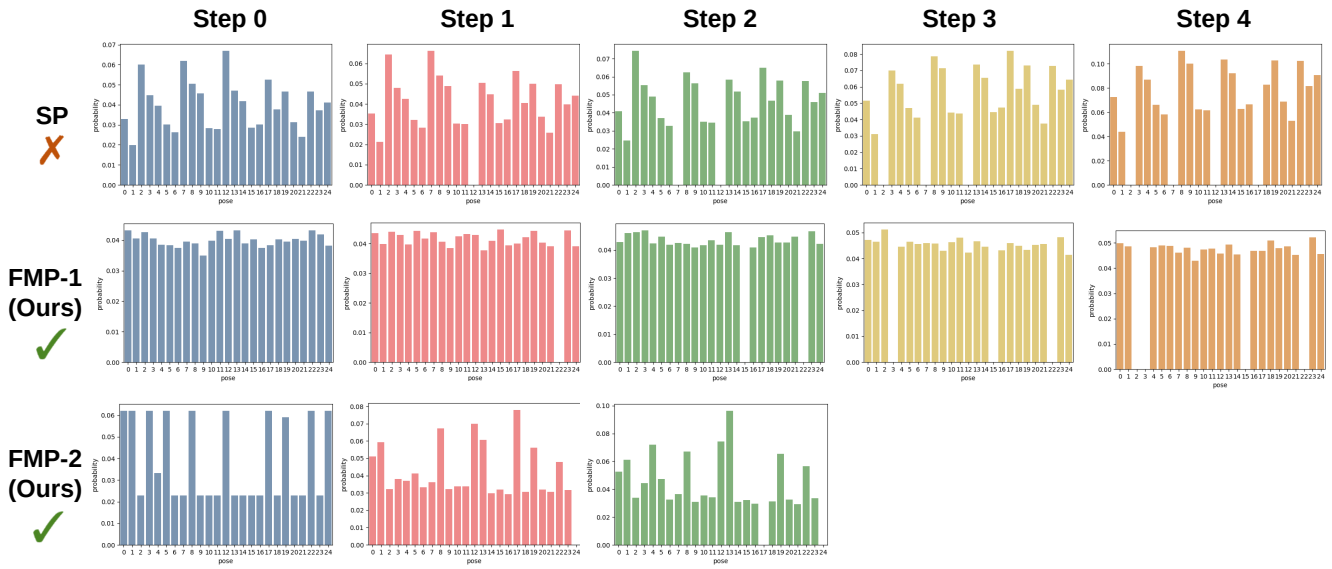


Fig. 12. Normalized probability distribution changes of the remaining actions at all trial steps of the case in Fig. 6. ✓means that the policy successfully find the right pose, while ✗means a failure. The decision sequences of three policies are **SP**: 12→7→2→17→8, **FMP-1**: 22→15→3→2→23, **FMP-2**: 24→17→13.

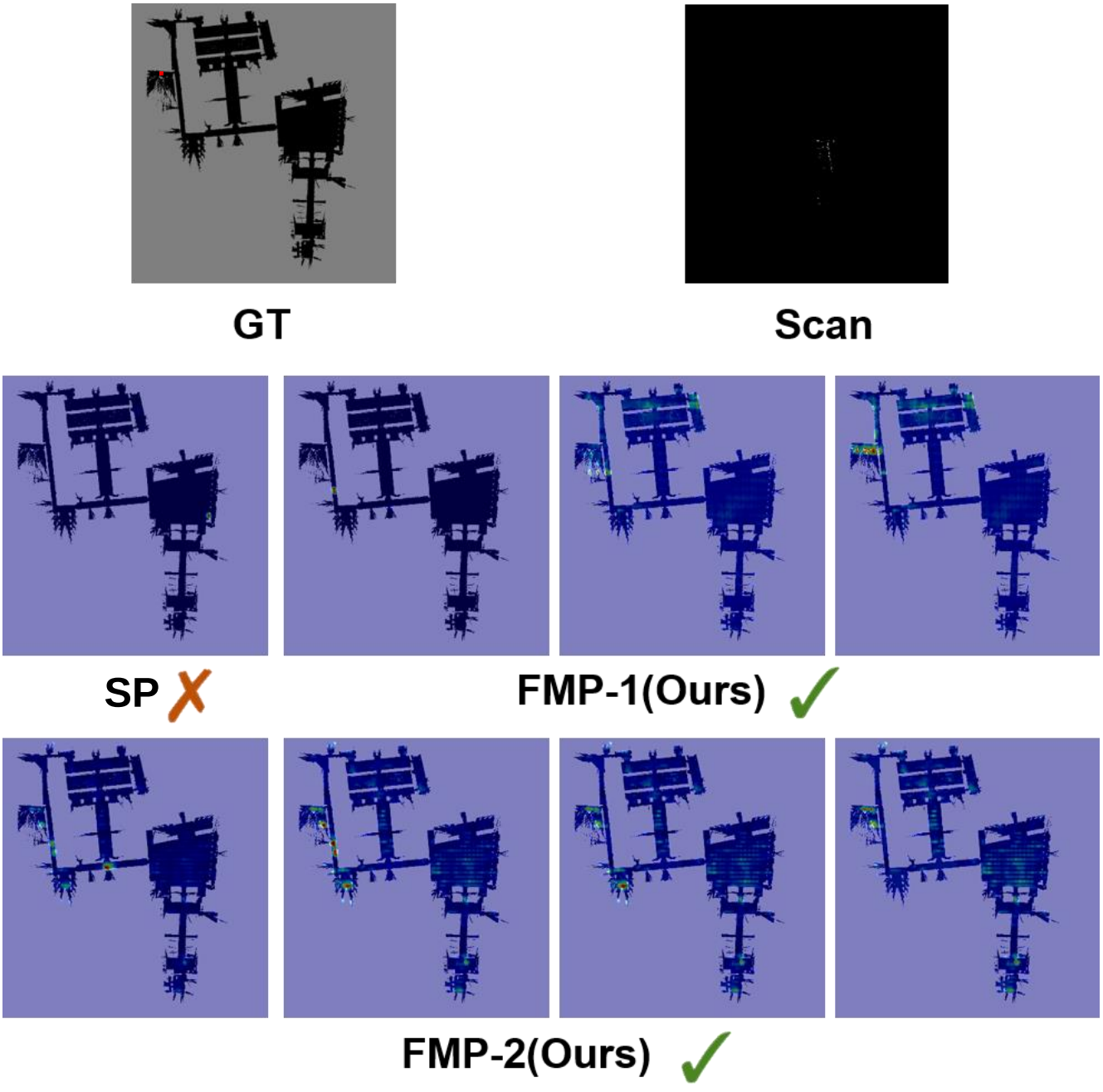


Fig. 13. A clear figure of Fig. 5.

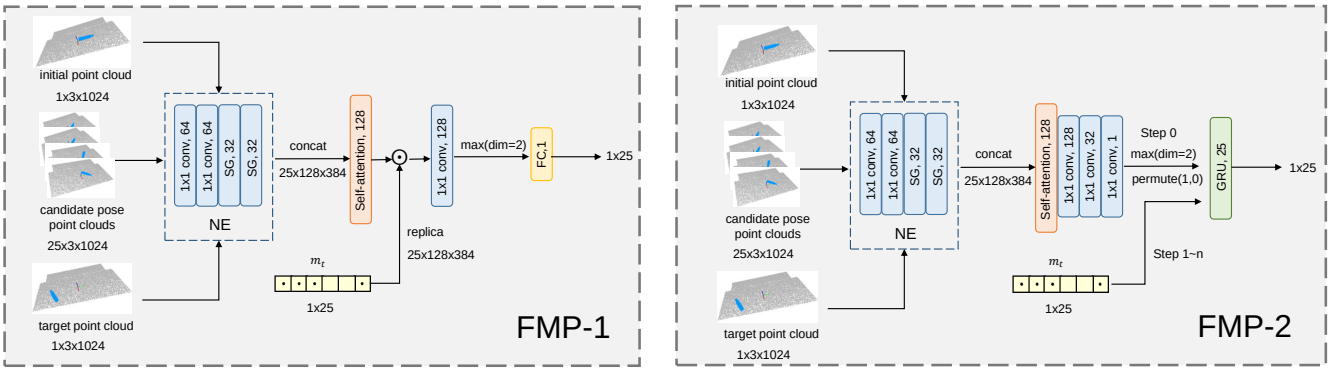


Fig. 14. Network architectures of reorientation task. Note that SG is sampling and grouping layers introduced in https://github.com/MenghaoGuo/PCT.

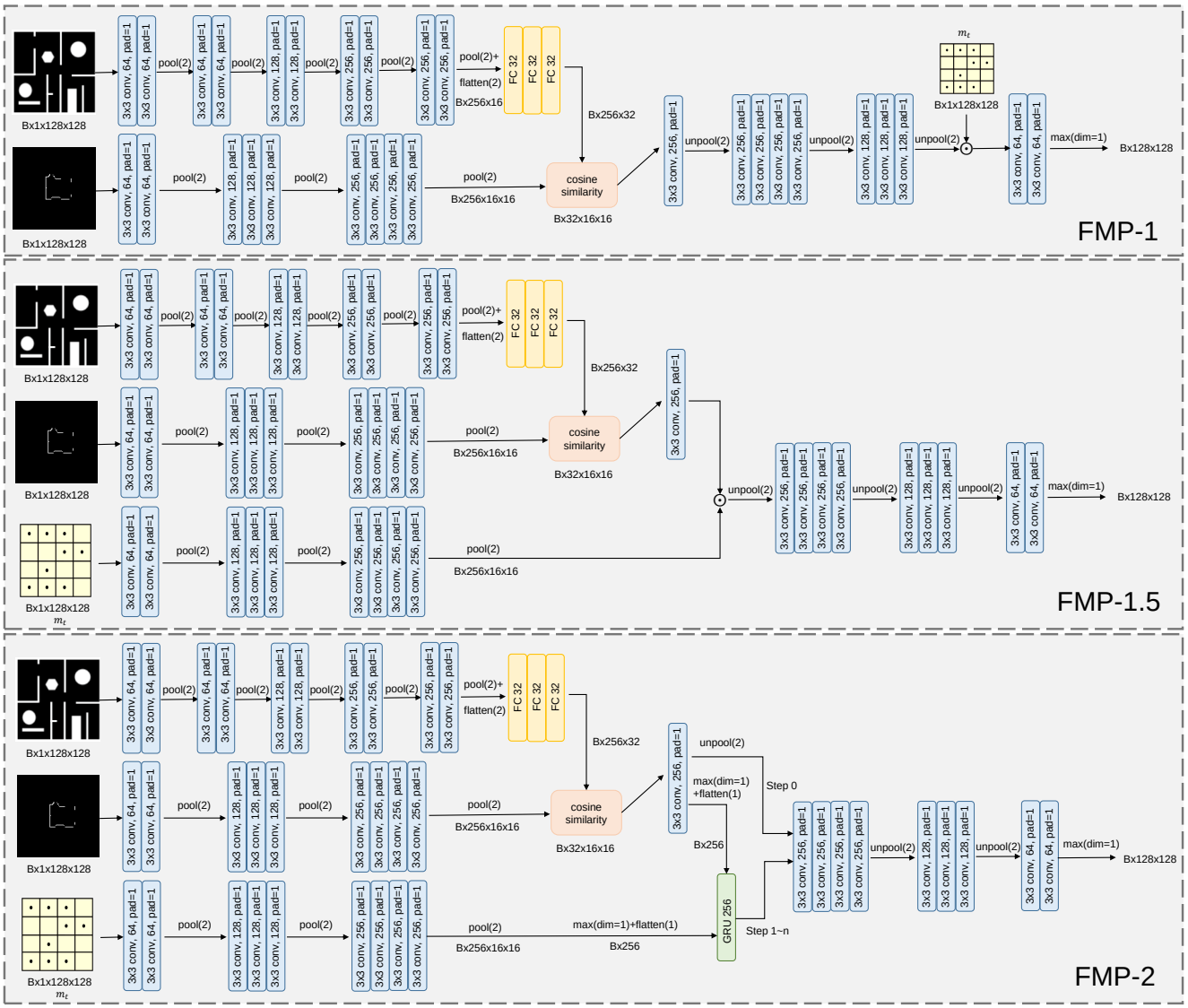


Fig. 15. Network architectures of localization task.