2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

ARiADNE: A Reinforcement learning approach using Attention-based Deep Networks for Exploration

Yuhong Cao[1], Tianxiang Hou[1], Yizhuo Wang[1], Xian Yi[1], Guillaume Sartoretti[1†]

***Abstract*— In autonomous robot exploration tasks, a mobile robot needs to actively explore and map an unknown environment as fast as possible. Since the environment is being revealed during exploration, the robot needs to frequently re-plan its path online, as new information is acquired by onboard sensors and used to update its partial map. While state-of-the-art exploration planners are frontier- and sampling-based, encouraged by the recent development in deep reinforcement learning (DRL), we propose ARiADNE, an attention-based neural approach to obtain real-time, non-myopic path planning for autonomous exploration. ARiADNE is able to learn dependencies at multiple spatial scales between areas of the agent's partial map, and implicitly predict potential gains associated with exploring those areas. This allows the agent to sequence movement actions that balance the natural trade-off between exploitation/refinement of the map in known areas and exploration of new areas. We experimentally demonstrate that our method outperforms both learning and non-learning state-of-the-art baselines in terms of average trajectory length to complete exploration in hundreds of simplified 2D indoor scenarios. We further validate our approach in high-fidelity Robot Operating System (ROS) simulations, where we consider a real sensor model and a realistic low-level motion controller, toward deployment on real robots.**

I. INTRODUCTION

Autonomous robot exploration (ARE) refers to the task, in which a robot needs to autonomously explore and map an unknown environment as efficiently and quickly as possible. The robot is usually equipped with a sensor (e.g., LiDAR or camera) to obtain measurements of its surroundings and build/update a *partial* map of the environment. In practice, the (high-dimensional) collected sensor data (e.g., point cloud) is usually converted into a (simplified) occupancy grid map or Octomap [1] that can be used for further planning [2], [3], [4]. Such a task is also known as active SLAM [2]. Although a robot can always slowly but accurately construct a map by carefully methodically covering the entire environment, the objective of ARE is to plan the shortest path to complete exploration, where small noises/errors in the final map are tolerable. In consequence, the main challenge of ARE is to plan a *non-myopic* path that balances the trade-off between exploiting surroundings (i.e., refining the map in already-explored areas) and exploring new (usually, further away) areas, most importantly with only partial knowledge of the environment. Such an exploration path is usually planned incrementally online, as the partial map is updated using new measurements along the way.

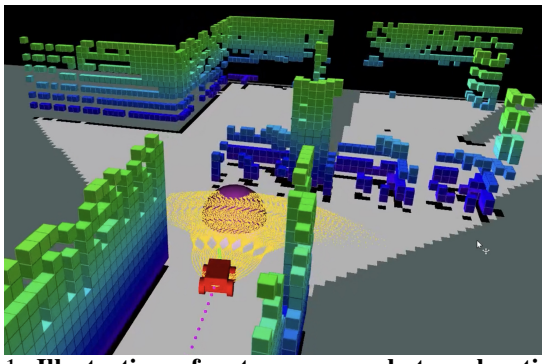


Fig. 1: **Illustration of autonomous robot exploration.** A wheeled robot is building a 3D Octomap using an onboard LiDAR. The purple ball indicates the next viewpoint output by our planner, which is tracked by the robot using a low-level motion controller (yellow primitives).

For example, conventional frontier-based methods [3], [5], [6], [7] generate multiple candidate paths, each covering a *frontier* (i.e., the boundary between explored free area and unexplored area), and greedily select the path with maximum *gain*, usually defined as a combination of *utility* (i.e., number of observable frontiers along the path) and *cost* (i.e., path length). However, an essential problem of these methods is that such myopic frontier selection does not guarantee optimality in the long term. A more recent approach [4] reasons about the whole path to cover all current frontiers, thus guaranteeing (near-)optimal paths in the current partial map. However, since the environment is only partially known, a previous optimal path often quickly becomes sub-optimal as more of the environment is revealed, or even worse, results in redundant movements (e.g., missing an unexplored shortcut between two rooms which were previously not known to be connected). Based on experiences from conventional exploration planners, we note that *non-myopicity* comes at two different levels in autonomous exploration. The first level, *spatial non-myopicity*, requires the planner to reason about the current partial map to balance the exploration-exploitation trade-off, while *temporal non-myopicity* requires the planner to estimate the future influence of current decisions (e.g., predict the changes in the partial map that may stem from given path planning decisions).

To achieve non-myopicity at these two levels, we propose a deep reinforcement learning (DRL) based approach for ARE, named ARiADNE, which relies on two attention-based neural networks. These networks allow the agent to reason

† Corresponding author, to whom correspondence should be addressed.
[1] Authors are with the Department of Mechanical Engineering, College of Design and Engineering, National University of Singapore. caoyuhong@u.nus.edu, htx24@foxmail.com, {wy98,yxian11}@u.nus.edu, mpegas@nus.edu.sg

about dependencies of different areas in the partial map at different spatial scales, thus allowing the agent to sequence spatially non-myopic decisions efficiently without the need to optimize a long path. Furthermore, our critic network implicitly provides the robot with the ability to estimate potential areas that might be found by learning the state-action value, which furthermore helps make decisions beneficial to the long-term efficiency, thus addressing temporally non-myopicity. Specifically, we first formulate autonomous exploration as a sequential decision-making problem on a collision-free graph that covers the known traversable area, where one of the nodes is the robot's current position. We then use our attention-based neural network to select one neighboring node of the current robot position as the next viewpoint for the robot. In this work, we focus on training and testing our approach in indoor environments based on 2D occupancy grid maps, ranging from simple scenarios (single room) to relatively complex ones (multiple rooms with complicated corridors). There, we experimentally demonstrate that our approach outperforms state-of-the-art conventional methods on average. We also validate our approach in high-fidelity ROS (Robot Operating System) simulations, where we consider a real sensor model and a motion controller, showing the generalizability of our model to realistic environments.

II. RELATED WORK

Frontier-based vs sampling-based approaches The first frontier-based method was proposed by Yamauchi [5], in which the robot is constantly driven towards the nearest frontier. In more advanced frontier-based methods, selections of which frontier to visit are evaluated by a gain function, which considers the effect of both utility and cost [6], [8], [9]. On the other hand, the past few years have seen a number of sampling-based methods be developed, based on Rapidly-exploring Random Trees (RRT) [3], Rapidly-exploring Random Graphs [10], and Probabilistic Random Maps (PRM) [11]. Sampling-based methods only need to compute the gain of sampled paths, which avoids the complexity of identifying and evaluating all frontiers. Recent works demonstrated that frontier-based methods are more suitable when frontiers are *sparse* (e.g., 2D indoor scenarios), while sampling-based methods perform better with *dense* frontiers (e.g., 3D outdoor scenarios) [7], [11]. Intuitively, Frontier-based methods become inefficient when there are too many frontiers to evaluate, while sampling-based methods underperform when informative paths are hard to sample.

Planning for long-term objectives Both frontier-based and sampling-based methods mostly rely on greedy strategies to plan short-term paths. Since the robot only has access to a partial map of the environment, such paths inevitably lead to myopic performance in the long term. Recently, Cao et al. [4] proposed TARE to optimize the full exploration path and mitigate myopicity. Utilizing the full knowledge of the current partial map, TARE was shown to significantly outperform state-of-the-art sampling-based planners in large-scale 3D scenarios. Similar methods were also considered to approach the *informative path planning* [12] problem, which considers information gathering usually in obstacle-free environments (in fact, works on autonomous exploration and informative path planning mutually promote each other, e.g., [4] and [12], [13] and [3]).

Learning-based exploration Niroui et al. [14] first combined frontier-based method with deep reinforcement learning to adaptively tune the parameter of the gain function for frontier selections and improve performance. Schmid et al. [15] proposed to learn the underlying gain distribution based on the partial map by supervised learning, thus helping sampling-based methods more efficiently find next-best-views and reduce computation. While the above works focus on improving conventional methods using machine learning, some other works [16], [17], [18], [19] directly applied deep reinforcement learning to select a viewpoint to visit, often relying on convolutional neural networks (CNNs). However, although [16], [18] argue that DRL-based methods naturally optimize long-term objectives, it seems that [16], [17], [18], [19] are only able to reach performance slightly better than the nearest-frontier method so far.

III. PROBLEM FORMULATION

We consider a bounded and unknown environment $\mathcal{E}$ represented by a $x \times y$ 2D occupancy grid map, whose partial (occupancy grid) map is denoted as P. The partial map consists of unknown area P_u (i.e., unexplored area) and known area P_k (i.e., explored area), such that $P_u \cup P_k = P$. The known area P_k is further classified into free area P_f (i.e., traversable area for the robot) and occupied area P_o (i.e., obstacles) such that $P_f \cup P_o = P_k$. At the beginning of exploration, the environment is fully unknown so the partial map $P = P_u$. Then, during exploration, the unknown area in the sensor range d_s (the sensor we use is a 360-degree LiDAR) is classified into either free area or occupied area according to sensor measurements. The objective of autonomous exploration is to find the shortest collision-free robot trajectory ψ^* to complete exploration:

$$\psi^* = \underset{\psi \in \Psi}{\operatorname{argmin}} \mathrm{C}(\psi), \ \text{s.t.}\ P_k = P_g, \tag{1}$$

where $\mathrm{C} : \psi \to \mathbb{R}^+$ maps a trajectory to its length and P_g denotes the ground truth of the environment. Although the ground truth is not accessible in real-world deployments, it is known and can be utilized to evaluate the performance of planners in testing. In practice, most works consider the closure of occupied areas as $P_k = P_g$ [5], [4], [18], [19].

IV. METHOD

In this section, we cast ARE as an RL problem, and introduce our attention-based policy and critic neural networks as well as details of our training.

A. Exploration as an RL Problem

Sequential Decision-making Problem Since the free area is updated based on the robot's movements, online planning for ARE is a sequential decision-making problem in nature.

Following our previous work [20] for informative path planning, we consider the robot trajectory ψ as a sequence of viewpoints $\psi = (\psi_0, \psi_1, ...), \psi_i \in P_f$. At each decision step t, we first uniformly distribute candidate viewpoints $V_t = \{v_0, v_1, ...\}, \forall\ v_i = (x_i, y_i) \in P_f$ in the current free area P_f, similar to [4]. Then, to find collision-free paths between viewpoints, we connect each viewpoint with its k nearest neighbors through a straight line and remove edges that collide with occupied or unknown areas. In doing so, we build a collision-free graph $G_t = (V_t, E_t)$, with V_t a set of uniformly distributed nodes (i.e., viewpoints) over the free area, and E_t a set of traversable edges. We finally let the robot select one neighboring node of its current position ψ_t as the next viewpoint. Since the decision will be taken upon arriving at the last selected viewpoint, the trajectory is a sequence of waypoints such that $\psi_i \in V$.

Observation The observation of the agent is $o_t = (G'_t, \psi_t)$, where $G'_t = (V'_t, E_t)$ is the *augmented graph* based on the current collision-free graph G_t, while ψ_t is the robot current position. Note that G'_t shares the same edge set E_t as G_t. In addition to the node coordinates (i.e., $v_i = (x_i, y_i)$), The properties of each node v'_i in the augmented graph further include a binary signal b_i, which indicates if the node has been visited by the agent already, and the associated utility u_i, such that $v'_i = (x_i, y_i, u_i, b_i)$. We experimentally found that the binary signal helps improve the learning by allowing the robot to be aware of its previous movements. The utility u_i represents the number of observable frontiers at node v_i [4]. We consider observable frontiers as frontiers within *light of sight* of the node (i.e., lines between the node and observable frontiers are collision-free and their length is smaller than the sensor range). The utility u_i at node v_i is computed as $u_i = |F_{o,i}|, \forall f_j \in F_{o,i}, ||f_j - v_i|| \leq d_s, L(v_i, f_j) \cap (P - P_f) = \emptyset$, where $F_{o,i}$ denotes the observable frontiers set at node v_i, d_s denotes the sensor range and $L(v_i, f_j)$ the line between node v_i and frontier f_j. In practice, we scale the node coordinates and utility to $[0, 1]$ before feeding the observation into the neural network.

Action At each decision step t, given the agent's observation o_t, our attention-based neural network outputs a stochastic policy to select a node out of all neighboring nodes as the next viewpoint to visit. The policy is denoted as $\pi_\theta(a_t|o_t) = \pi_\theta(\psi_{t+1} = v_i, (\psi_t, v_i) \in E_t \mid o_t)$, where θ represents the set of weights of the neural network. The robot moves to the next viewpoint in a straight line, and updates its partial map based on data collected along the way.

Reward To encourage efficient exploration, after taking each movement action a_t, the robot receives a reward composed of three parts. The first part $r_o = |F_{o,\psi_{t+1}}|$ is the number of observed frontiers at the new viewpoint. The second part $r_c = -\mathrm{C}(\psi_t, \psi_{t+1})$ is a punishment on the distance between the previous and new viewpoints. A fixed finishing reward $r_f = \begin{cases} 20, & P_k = P_g \\ 0, & \text{otherwise,} \end{cases}$ is given at the end of the episode, if and only if the exploration task was completed. The total reward reads: $r_t(o_t, a_t) = a \cdot r_o + b \cdot r_c + r_f$, where a and b are scaling parameters (in practice $a = 1/50, b = 1/64$).

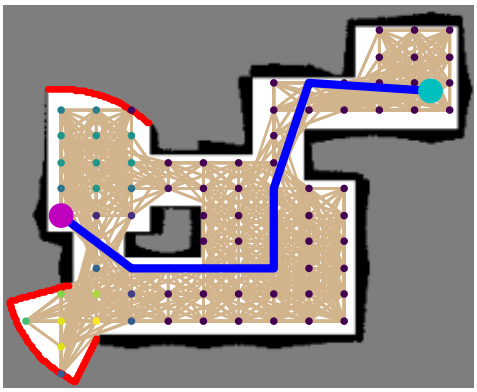


Fig. 2: **Example decision step in the middle of an exploration task in our approach**, showing the unknown area (grey cells), free area (white cells), occupied area (black cells), frontiers (red cells), executed trajectory (blue line), graph edges (tan lines), candidate viewpoints (small dots, whose color represents their utility), robot current position (purple disk), and robot starting position (light blue disk).

B. Policy Network

The policy ψ_θ is output by our attention-based neural network, which is composed of an encoder and a decoder (shown in Fig. 3). We first rely on the encoder to extract salient features from the current partial map, specifically by learning dependencies between nodes in the associated augmented graph G'. Based on these features as well as the current robot position, the decoder then outputs the policy over neighboring nodes, which can be used to decide which one to visit next. Note that, while policy-based RL agents often have a fixed action space, our decoder is inspired by the Pointer Network [22] to allow the action space to depend on the number of neighboring nodes input in the network. This allows our network to naturally adapt to our collision-free graph, where nodes have arbitrary numbers of neighbors.

Attention Layer We use the attention layer [21] as the fundamental building block in our model. The input of such an attention layer is composed of a query vector h^q and a key-and-value vector $h^{k,v}$. The output of this layer, h'_i, is the weighted sum of the value vector, where weights depend on the similarity between key and query:

$$q_i = W^Q h_i^q,\ k_i = W^K h_i^{k,v},\ v_i = W^V h_i^{k,v},$$
$$u_{ij} = \frac{q_i^T \cdot k_j}{\sqrt{d}},\ w_{ij} = \frac{e^{u_{ij}}}{\sum_{j=1}^{n} e^{u_{ij}}},\ h'_i = \sum_{j=1}^{n} w_{ij} v_j, \tag{2}$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d \times d}$ are all learnable matrices. Updated features are then passed through a feed-forward sublayer, following [21].

Encoder In the encoder, we first linearly embed the *node inputs* V' into d-dimensional *node features* h^n, where $h_i^n = W^l v'_i + b^l$. We then calculate an edge mask M where $m_{ij} = \begin{cases} 0, & (v_i, v_j) \in E_t \\ 1, & (v_i, v_j) \notin E_t \end{cases}$. The node features are then passed to

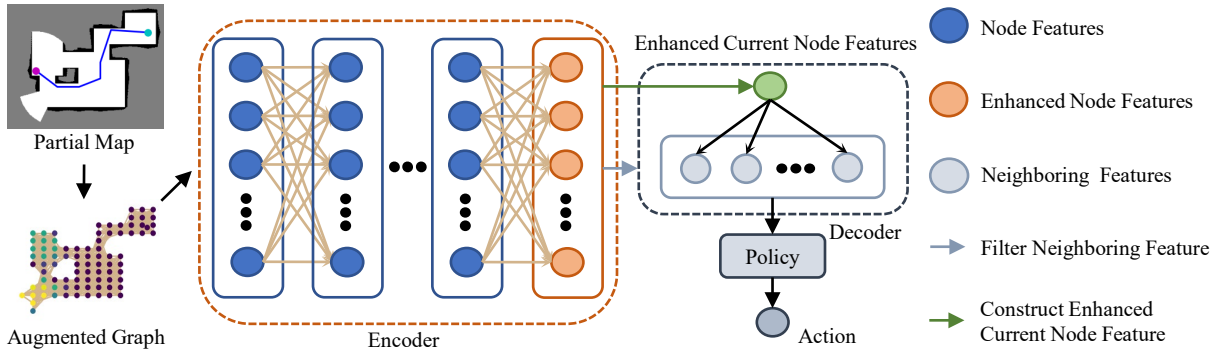


Fig. 3: **Attention-based policy network.** Note that neighboring relationships in the augmented graph (tan) are also used as the mask [21] in attention layers in the encoder.

multiple (6 in practice) stacked attention layers, where $h^q = h^{k,v} = h^n$, each attention layer taking the output of the previous one as input. An edge mask is applied to allow each node access to its neighboring node features only, by setting $w_{ij} = 0,\ \forall(i,j),\ m_{ij} = 1$. Despite attention being restricted to neighboring nodes in each layer, nodes can still obtain non-neighboring node features by aggregating node features multiple times through this stacked attention structure. We empirically found that such structure is more suitable than graph transformers [23] (like in our previous work [20]) to learn path finding in maps with cluttered obstacles. We term the output of the encoder, $\hat{h}^e$, the *enhanced node features*, since each of these updated node features $\hat{h}^n_i$ contains the dependencies of v'_i with other nodes.

Decoder We use the decoder to output a policy based on enhanced node features $\hat{h}^e$ and the current robot position ψ_t. Denoting the current robot position as node $v_c = \psi_t$, we first select the *current node features* h^c and *neighboring features* $h^{nb},\ \forall \hat{h}^{nb}_i,\ (v_c, v_i) \in E_t$ from the corresponding enhanced node features. We then pass the current node features and enhanced node features to an attention layer, where $h^q = h^c, h^{k,v} = \hat{h}^n$, concatenate its output with h^c, and project it back to a d-dimensional feature vector. We term this vector the *enhanced current node features* $\hat{h}^c$. After that, we pass the enhanced current node features and neighboring features to a pointer layer [22], an attention layer directly outputting the attention weights w as the output with $h^q = \hat{h^c},\ h^{k,v} = h^{nb}$. We finally take the output of this pointer layer as the robot's policy, i.e., $\pi_\theta(a_t \mid o_t) = w_i$.

C. Critic Network

We train the policy network using the soft actor critic (SAC) algorithm [24], [25] (see details below), where a critic network is trained to predict state-action values. Since state-action values approximate long-term *returns* (the accumulated sum of rewards), we believe that they also implicitly predict potential gains (i.e., potential areas that might be found), which further helps the robot sequence non-myopic decisions. In practice, we train a critic network to approximate soft state-action values $Q_\phi(o_t, a_t)$, where ϕ denotes the set of weights of the critic network. The structure of the critic network is nearly the same as the policy network, except that there is no pointer layer at the end of the decoder. Instead, we directly concatenate the enhanced current node features and neighboring features, then project them to soft state-action values.

D. Training

Soft Actor-critic SAC aims to learn a policy that maximizes *return* while keeping its entropy as high as possible:

$$\pi^* = \text{argmax}\, \mathbb{E}_{(o_t,a_t)}[\sum_{t=0}^{T} \gamma^t (r_t + \alpha \mathcal{H}(\pi(.|o_t)))], \qquad (3)$$

where π^* is the optimal policy, T the number of decision steps, γ the discount factor, and α the temperature parameter that tunes the importance of the entropy term versus the return. In SAC, the soft state value is calculated as: $V(o_t) = \mathbb{E}_{a_t}[Q(o_t, a_t)] - \alpha \log(\pi(a_t|o_t))$.
The critic loss is calculated as: $J_Q(\phi) = \mathbb{E}_{o_t}[\frac{1}{2}(Q_\phi(o_t, a_t) - (r_t + \gamma \mathbb{E}_{o_{t+1}}[V(o_{t+1})]))^2]$.
The policy loss loss is calculated as: $J_\pi(\theta) = \mathbb{E}_{(o_t,a_t)}[\alpha \log(\pi_\theta(a_t|o_t)) - Q_\phi(o_t, a_t)]$.
The temperature parameter is auto-tuned during the training and the temperature loss is calculated as: $J(\alpha) = \mathbb{E}_{a_t}[-\alpha(\log \pi_t(a_t|o_t) + \overline{\mathcal{H}})]$, where $\overline{\mathcal{H}}$ denotes the target entropy. In practice, we use double target networks for the critic network training, as in [24], [25].

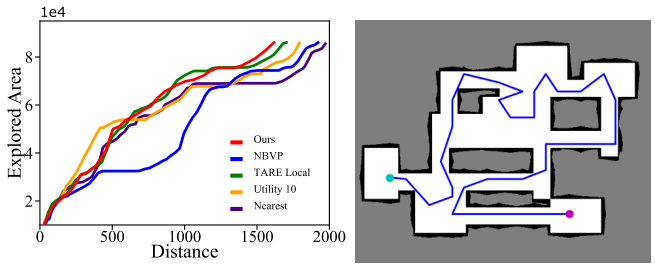
Training Details We utilize the same environments provided in [18] for training, which are generated by a random dungeon generator. Each environment is a 640×480 grid map, while the sensor range $d_s = 80$. To build the collision-free graph, 900 points are uniformly distributed to cover the whole environment, with all points in the known free area considered as candidate viewpoints V. We check the $k = 20$ nearest neighbor of each viewpoint, and connect them if such an edge is collision-free, to form the edge set E. We consider the exploration task to be completed once more than 99% of the ground truth has been explored ($|P_k|/|P_g| > 0.99$). During training, we set the max episode length to 128 decision steps, the discount factor to $\gamma = 1$, the batch size to 256, and the episode buffer size to $10,000$. Training starts after the episode buffer collects more than 2000 steps data. The target entropy is set to $0.01 \cdot \log(k)$. Each training step contains 1 iteration and happens after 1 episode finishes. We use the Adam optimizer with a learning rate of 10^{-5} for both policy and critic networks. The target

TABLE I: **Comparison with baseline ARE planners (100 scenarios for each test set).** We report the average and standard deviation of the trajectory length to complete exploration (lower is better). For utility-based methods [6], the numbers 1, 10, 25 represent the value of λ, which is used to tune exploitation and exploration.

	Nearest	Utility 1	Utility 10	Utility 25	NBVP	TARE Local	CNN	ARiADNE
easy	772(±253)	736(±266)	732(±256)	764(±258)	745(±268)	692(±228)	779(±281)	**663**(±257)
medium	1248(±295)	1266(±311)	1179(±300)	1227(±307)	1217(±271)	1170(±275)	1169(±319)	**1130**(±334)
complex	1669(±332)	1873(±457)	1662(±347)	1711(±352)	1744(±366)	1646(±312)	1647(±422)	**1599**(±363)
random	1354(±410)	1423(±466)	1268(±396)	1315(±413)	1323(±371)	1266(±388)	1323(±428)	**1204**(±378)

(a) simple (b) medium (c) complex

Fig. 4: **Examples scenarios from each different test set.**



(a) Trajectory Analysis (b) ARiADNE (1618)

(c) TARE Local (1703) (d) NBVP (1922) (e) Utility 10 (1793)

Fig. 5: **Visual comparison of our method and baselines in an example scenario.**

critic network updates every 256 training steps. Our model is trained on a workstation equipped with a i9-10980XE CPU and an NVIDIA GeForce RTX 3090 GPU. We train our model utilizing Ray, a distributed framework for machine learning [26], to parallelize and accelerate data collection (32 instances in practice). The training needs around 24h to converge. Our full training and testing code is available at `https://github.com/marmotlab/ARiADNE`.

V. EXPERIMENT

A. Comparison Analysis

Most previous works often only conduct experiments in a few scenarios (often less than 10). However, we note that the performance of exploration planners exhibits high variance in different scenarios. Therefore, we believe a convincing comparison should be based on evaluation in a large number of testing environments. Although building so many testing environments is tricky and time-consuming even in ROS, hundreds of simplified scenarios, like the ones we used for training, can be generated easily. Therefore, we conduct comparison analyses on a fixed set of simplified environments, which were never seen by our trained model. Testing environments are divided in four sets (100 scenarios each), named *random*, *easy*, *medium*, and *complex*. *Easy* scenarios only contain one room, and *complex* scenarios contain multiple rooms with complicated corridors, while the complexity of *medium* scenarios lies in-between. *Random* scenarios contain a mix of *easy*, *medium*, and *complex* scenarios (but no repeated scenario from these test sets).

We compare ARiADNE with state-of-the-art conventional planners, including Nearest Frontier [5], Utility-based Frontier [6], NBVP [3], and TARE Local [4]. Nearest Frontier always drives the robot towards the nearest frontier, while Utility-based Frontier evaluates the gain of each frontier $g_i = u_i \cdot e^{-\lambda \cdot C(\psi_i)}$ and drives the robot to the frontier with the highest gain, where u_i is the utility of frontier i, ψ_i the shortest path from the robot's current position to frontier i, and λ a tunable parameter used to balance exploration and exploitation. The same function is also used in NBVP to evaluate sampled trajectories. We tried a series of values of λ for Utility-based Frontier and NBVP, and found that $\lambda = 10$ generally performs best (see Table I). Finally, TARE Local refers to the local planner of TARE [4], which explicitly plans a full trajectory to cover all frontiers (we do not use TARE's global planner, since its local planning horizon already fits our testing environments). NBVP and TARE run 300 and 10 iterations for each decision step respectively (15 and 1 in default [3], [4]), to make their decisions as optimal as possible. In our tests, we adopt our collision-free graph as the trajectory space for all baselines except NBVP (we found RRTs mostly generate poor zig-zag paths due to symmetries in our uniform graph), to alleviate the randomness of sampling and ensure a fair comparison. We further compare against a CNN-based DRL planner [18]. Since this CNN-based planner has a fixed observation range, it only has a partial observation of the (partial) map, and relies on a frontier-based method for exploration when there is no nearby frontier in its field-of-view.

We report the average and variance of the total trajectory length to complete exploration in Table I. Our results indicate that ARiADNE outperforms all baselines on average, in all test sets. We do not report the planning time of baseline methods in Table I, since we focused on implementing fundamental inner workings of NBVP and TARE Local, without perfectly optimizing their computing time (their per-step computing requires more than one second). Despite this, we note that under our exploration setting, the average per-step computing time of our approach is lower than 0.1s on a i7-1270P@2.2GHz CPU and thus can be used in real-time and fast reactive to the map change.

As discussed in the related work section, the best-tuned

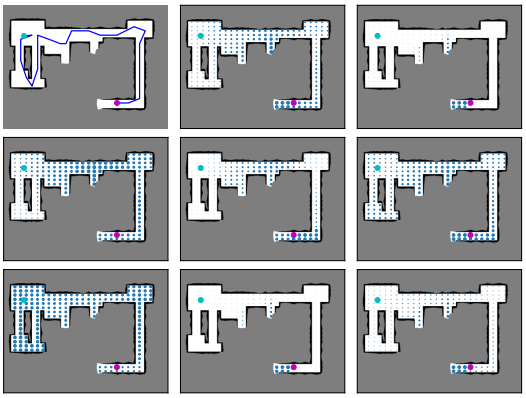
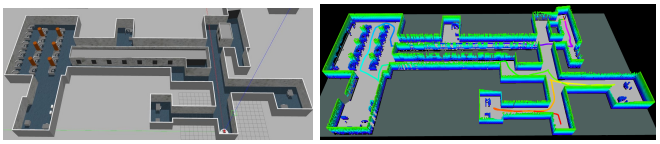
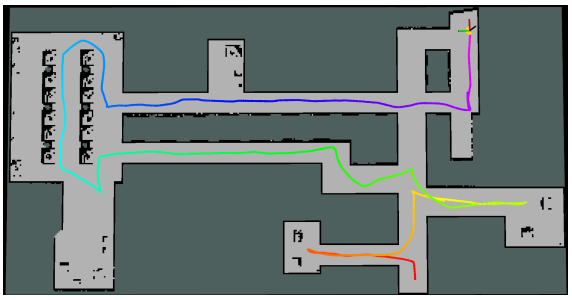


Fig. 6: **Attention weights visualization of the critic network decoder.** The query vector is the node at the current robot's position (purple) and the keys vector are nodes in the augmented graph (blue). Note how the different heads of the decoder learn to focus on either local or global dependencies of areas in the partial map.



(a) Ground truth (b) Constructed Octomap



(c) Constructed occupancy grid map

Fig. 7: **Validation of our method in simulation.** Note that ignoring the small left-down corner is actually a wise decision since the objective is to explore 99% of the environment.

frontier-based method (Utility 10) performs well in 2D exploration tasks (better than NBVP). Despite this, since these frontier-based methods are myopic, they are outperformed by TARE Local, which plans near-optimal long-term (full) trajectories on the current partial map. While it only constructs paths one viewpoint at a time, our learning-based method can not only reason about the whole partial map to construct efficient, non-myopic exploration trajectories, while learning to predict the potential long-term gain of decisions. We believe such an advantage results in the improvement of our method over conventional baselines (5% better than TARE Local in our *random* scenarios). Fig. 5 shows an example where ARiADNE plans a more efficient trajectory, while conventional methods suffer from redundant movements. However, it should be noted that considering long-term paths and predicting potential gains do not strictly guarantee better performance in every scenario (e.g., predictions could be wrong). In fact, ARiADNE plans the shortest path for 33 scenarios in our *random* tests, while TARE Local, NBVP, Utility 10 perform best in 23, 21, 23 scenarios respectively.

Finally, ARiADNE also outperforms the CNN-based planner. We believe that our main advantage stems from the attention-based neural network, which efficiently learns features at different scales (as shown in Fig. 6, different heads of the decoder learn to focus on either local or global dependencies), while CNNs naturally only focus on local dependencies. Therefore, our model can better learn dependencies between different areas to reason about the entire partial map and avoid myopic decisions.

B. Experimental validation

We validate ARiADNE in a simulation environment for exploration provided by [27]. It contains fundamental modules (e.g., state estimation and motion control), which allow us to consider a real sensor model and a low-level motion controller. The validation is conducted in a realistic indoor environment (approximately 70m $\times$ 40m) with long and narrow corridors connected with tables, colums, and lobby areas (see Fig. 7(a)). We use a wheeled robot equipped with a 3D Velodyne Lidar with a 130m sensor range. We convert collected data into an Octomap (see Fig. 7(b)) and then project it to a occupancy grid map for exploration planning (see Fig. 7(c)). The resolution of the grid map is 0.2m. We re-plan the path every 0.2s. Although the sensor model (i.e., sensor range and sensing frequency) of the robot is drastically different from the one used in training, our trained model still makes efficient decisions to avoid redundant movements for exploration (see the colored trajectory in Fig. 7(c), highlighting the generalizability of our approach.

VI. CONCLUSION

In this work, we propose ARiADNE, a reinforcement learning approach that relies on attention-based deep neural network for autonomous exploration. Our approach allows the robot to efficiently learn dependencies between different areas in its partial map and implicitly predict potential gains, thus allowing it to sequence non-myopic movement decisions in partially-known environments. In our tests, ARiADNE exhibits improvement over state-of-the-art frontier-based, sampling-based, and CNN-based exploration planners, in terms of average trajectory length to complete exploration. We also validate our approach in a high-fidelity ROS simulation, where we consider a real sensor model and a low-level motion controller, towards deployments on real robots.

Future work will focus on extending our approach to autonomous exploration of 3D environments, where frontiers are much denser than in 2D. Second, although in this work we uniformly distribute nodes to construct a graph, we believe a sparser graph containing more informative viewpoints may improve performance. Finally, we are also interested in explicitly predicting the potential gain during exploration to further boost planning performance.

ACKNOWLEDGMENTS

This work was supported by Temasek Laboratories (TL@NUS) under grant TL/FS/2022/01.

REFERENCES

[1] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at https://octomap.github.io. [Online]. Available: https://octomap.github.io

[2] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," *arXiv preprint arXiv:2207.00254*, 2022.

[3] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon" next-best-view" planner for 3d exploration," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1462–1468.

[4] C. Cao, H. Zhu, H. Choset, and J. Zhang, "Tare: A hierarchical framework for efficiently exploring complex 3d environments." in *Robotics: Science and Systems*, 2021.

[5] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*. IEEE, 1997, pp. 146–151.

[6] H. H. González-Banos and J.-C. Latombe, "Navigation strategies for exploring indoor environments," *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 829–848, 2002.

[7] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient autonomous exploration planning of large-scale 3-d environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1699–1706, 2019.

[8] D. Holz, N. Basilico, F. Amigoni, and S. Behnke, "Evaluating the efficiency of frontier-based exploration strategies," in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*. VDE, 2010, pp. 1–8.

[9] M. Kulich, J. Faigl, and L. Přeučil, "On distance utility in the exploration task," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4455–4460.

[10] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, "Graph-based subterranean exploration path planning using aerial and legged robots," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1363–1388, 2020.

[11] Z. Xu, D. Deng, and K. Shimada, "Autonomous uav exploration of dynamic environments via incremental sampling and probabilistic roadmap," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2729–2736, 2021.

[12] S. Arora and S. Scherer, "Randomized algorithm for informative path planning with budget constraints," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4997–5004.

[13] G. A. Hollinger and G. S. Sukhatme, "Sampling-based robotic information gathering algorithms," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1271–1287, 2014.

[14] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[15] L. Schmid, C. Ni, Y. Zhong, R. Siegwart, and O. Andersson, "Fast and compute-efficient sampling-based local exploration planning via distribution learning," *arXiv preprint arXiv:2202.13715*, 2022.

[16] D. Zhu, T. Li, D. Ho, C. Wang, and M. Q.-H. Meng, "Deep reinforcement learning supervised autonomous exploration in office environments," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 7548–7555.

[17] H. Li, Q. Zhang, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 6, pp. 2064–2076, 2019.

[18] F. Chen, S. Bai, T. Shan, and B. Englot, "Self-learning exploration and mapping for mobile robots via deep reinforcement learning," in *Aiaa scitech 2019 forum*, 2019, p. 0396.

[19] F. Chen, J. D. Martin, Y. Huang, J. Wang, and B. Englot, "Autonomous exploration under uncertainty via deep reinforcement learning on graphs," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6140–6147.

[20] Y. Cao, Y. Wang, A. Vashisth, H. Fan, and G. A. Sartoretti, "CAtNIPP: Context-Aware Attention-based Network for Informative Path Planning," in ***Accepted to*** *the 6th Annual Conference on Robot Learning*, 2022. [Online]. Available: https://openreview.net/forum?id=cAIIbdNAeNa

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[23] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.

[24] P. Christodoulou, "Soft actor-critic for discrete action settings," *arXiv preprint arXiv:1910.07207*, 2019.

[25] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[26] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, *et al.*, "Ray: A distributed framework for emerging ai applications," in *Proceedings of OSDI*, 2018, pp. 561–577.

[27] C. Cao, H. Zhu, F. Yang, Y. Xia, H. Choset, J. Oh, and J. Zhang, "Autonomous exploration development environment and the planning algorithms," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8921–8928.