DEEP REINFORCEMENT LEARNING FOR AUTONOMOUS ROBOT EXPLORATION UNDER UNCERTAINTY

by

Fanfei Chen

A DISSERTATION

Submitted to the Faculty of the Stevens Institute of Technology
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Fanfei Chen, Candidate

ADVISORY COMMITTEE

Brendan Englot, Chairman Date

Philippos Mordohai Date

Kishore Pochiraju Date

Long Wang Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2021

DEEP REINFORCEMENT LEARNING FOR AUTONOMOUS ROBOT EXPLORATION UNDER UNCERTAINTY

ABSTRACT

Mapping and exploration of a priori unknown environments are crucial capabilities for mobile robot autonomy. Information-theoretic exploration methods were developed to guide robots to the unexplored areas of their environment that will contribute the largest quantities of new information to an occupancy map. Meanwhile, simultaneous localization and mapping (SLAM) provides a way to evaluate map accuracy and manage localization error under noisy relative measurements. Consequently, SLAM-based exploration or active SLAM has been applied successfully for mapping an unknown environment autonomously under robot landmark and pose uncertainty. However, the decision-making component of active SLAM exploration methods is time-consuming due to the need for forward simulation of future robot measurements, and prediction of the resulting map and pose uncertainty. This approach will ultimately fail for real-time decision-making with increasing dimensionality of the state space and the action space, due to the costly time complexity of such methods.

In this proposed work, we present learning-based exploration algorithms to offer reduced computation time and near-optimal exploration strategies under uncertainty. First, we propose a method to solve autonomous mobile robot exploration using a robot's local map and deep reinforcement learning (DRL) without considering localization uncertainty. The DRL controller generates robot sensing actions that are nearly as informative and efficient as those of a standard mutual information maximizing controller, at a substantial reduction in computational effort during the online testing phase. Second, we present an approach that uses graph neural net-

works (GNNs) in conjunction with DRL by taking proposed exploration graphs as input data, enabling decision-making over graphs containing exploration information to predict a robot's optimal sensing action in belief space. Third, we demonstrate that the proposed GNN-based RL policy can be trained efficiently in a simulation environment and perform zero- shot RL transfer to both virtual and real environments containing different obstacle quantities, arrangements, and geometries without parameter-tuning. Fourth, we propose to learn an exploration policy from a portfolio of algorithms that are well-suited to different situations, such that the learned policy is capable of choosing an optimal exploration strategy according to the current state of the environment.

Author: Fanfei Chen

Advisor: Brendan Englot

Date: August 20, 2021

Department: Mechanical Engineering

Degree: Doctor of Philosophy

To my family and my parents.

Acknowledgments

First and foremost I would like to express my deepest appreciation to my advisor Dr. Brendan Englot, for support and patience during my Ph.D. study in the Robust Field Autonomy Lab (RFAL). Dr. Englot's immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. I am especially grateful that he recognized my potential during my master's studies and recruited me. I appreciate the valuable time he spent on our weekly one-on-one meetings to guide my research. Dr. Englot's positive attitude and growth mindset tremendously promote creativity and free discussion in our lab. Without his guidance and support, this dissertation would not have been possible.

I would also like to show gratitude to my committee, Dr. Philippos Mordohai, Dr. Kishore Pochiraju, and Dr. Long Wang for their encouragement, insightful comments, and detailed feedback.

My sincere thanks also goes to my fellow labmates: Dr. Shi Bai, Dr. Tixiao Shan, Dr. Jinkun Wang, Kevin Doherty, Dr. John Martin, Erik Pearson, Dr. Dengwei Gao, Sumukh Patil, Dong Cui, Jake McConnell, Yewei Huang, and Paul Szenher. I have received many helpful suggestions from discussion with you and I will remember the cherished time spent together in the lab. I want to thank Dr. Mishah Salman for all the advice for my research, and it has been a great pleasure to be his teaching assistant. I want to thank Jennifer Field for her help with my TA duty. I want to thank Chenhui Zhao for being a good friend and lunch buddy at Stevens.

Finally, I would like to express my gratitude to my parents for their emotional and financial support. My research work couldn't be done without their support. I want to thank my wife Wenqian Zhang for her enduring support of me during my doctoral work and bringing our beautiful daughter Thea to the world.

Funding Acknowledgement: This thesis was supported in part by the National Science Foundation (NSF), grant number IIS-1723996.

Table of Contents

Chapter 1
Introduction

1.1 Motivation and Problem Statement

It is challenging to solve an autonomous mobile robot exploration problem in an *a priori* unknown environment when localization uncertainty is a factor, which may compromise the accuracy of the resulting map. Due to our limited ability to plan ahead in an unknown environment, a key approach to solving the problem is often to estimate and select the immediate next best viewpoint. The next-view candidates may be generated by enumerating frontier locations or using sampling-based methods to plan beyond frontiers. A utility function may be used to evaluate the next-view candidates and select the optimal candidate. It is common to forward-simulate the actions and measurements of each candidate and choose the best one as the next-view position. However, the decision-making component of active SLAM exploration methods is time-consuming due to the need for forward simulation of future robot measurements, and prediction of the resulting map and pose uncertainty. This approach will ultimately fail for real-time decision-making with increasing dimensionality of the state space and the action space, due to the costly time complexity of such methods.

Learning-based exploration algorithms can accurately predict optimal next-view positions with nearly constant computation time by taking camera image or occupancy grid map as input data. However, an occupancy grid map has limitations as an input to represent the environment. For example, in some instances neural networks may be unable to perform reliably over a map with a rotation relative to

the maps presented in training. Hence, such a framework may take a longer time to converge during the training phase, or even fail to converge, due to the size of the state space induced by an occupancy map.

The goal of this dissertation is to design a learning-based exploration algorithm which can efficiently and robustly achieve sim2sim and sim2real transfer. We aim to transfer a learned exploration policy from the training environment to the test environment without parameter-tuning.

1.2 Overview and Contributions

This dissertation is organized as follows. In Chapter 2, we review the methodologies that are proposed for achieving autonomous exploration. These methodologies cover exploration, SLAM and Expectation-Maximization Exploration.

In subsequent chapters, the main contributions of this dissertation are discussed as follows:

- A novel method to solve autonomous mobile robot exploration using a robot's local map and deep reinforcement learning (DRL) without considering localization uncertainty [1] (Chapter 3);
- A novel approach that uses graph neural networks (GNNs) by taking proposed exploration graphs as input data, enabling decision-making over graphs containing exploration information to predict a robot's optimal sensing action in belief space [2], [3] (Chapter 4);
- A novel zero-shot RL transfer framework that can learn exploration policies efficiently in a simulation environment and perform policy transfer to both virtual and real environments without parameter-tuning [4] (Chapter 5).

Chapter 2
Background

2.1 Simultaneous Localization and Mapping (SLAM)

2.1.1 Graph-based SLAM

A graph-based SLAM framework [5], [6] solves the estimation problem by performing incremental nonlinear least-squares smoothing over a factor graph that represents the current state. A factor graph is a bipartite graph containing factor nodes and variable nodes. The variable nodes are unknown variables in the estimation problem. The factor nodes are functions of related variables. Edges in the factor graph indicate the dependencies of the factor node and the corresponding variable node so they are always between factors and variables.

We consider the trajectory by a set of variables $\mathbf{x}_{1:T} = \{x_1, x_2, ..., x_T\}$. The control input variables are denoted as $\mathbf{u}_{1:T} = \{u_1, u_2, ..., u_T\}$ and the measurements are represented as $\mathbf{z}_{1:T} = \{z_1, z_2, ..., z_T\}$. The goal of the SLAM problem is to solve for the posterior probability:

$$p(\mathbf{x}_{1:T}, m|\mathbf{z}_{1:T}, \mathbf{u}_{1:T}, x_0), \tag{2.1}$$

where m is the map and x_0 is an initial position. We also define virtual observation measurements $\hat{z}(x_i, x_j)$, which represent an observation of a transformation from position x_i to position x_j. The measurement error between the real observation and the virtual observation is defined as:

$$e_{ij}(x_i, x_j) = z_{ij} - \hat{z}_{ij}(x_i, x_j). \tag{2.2}$$

The log-likelihood l_{ij} of the observation difference associated with the measurement z_{ij} is:

$$l_{ij} \propto e_{ij}^T \Omega_{ij} e_{ij}, \tag{2.3}$$

where Ω_{ij} is the information matrix of the measurement between positions i and j. We wish to recover the estimated trajectory that minimizes the measurement error as follows:

$$x^* = \arg\min_x \sum_{i,j} e_{ij}^T \Omega_{ij} e_{ij}. \tag{2.4}$$

This nonlinear least-squares problem can be solved by various optimization methods. The optimization step can be achieved in real-time because of the sparsity of the information matrix [7].

2.1.2 Active SLAM

Active SLAM exploration approaches have been developed to reduce both the uncertainty of a robot's pose and entropy of the map by considering the correlation between localization and information gain [8]. A typical active SLAM algorithm gradually populates an incomplete stochastic map of the environment. At each decision-making step, i candidate trajectories are evaluated with respect to a cost function containing an uncertainty criterion [9]. The underlying SLAM algorithm is used to predict the posterior map resulting from each candidate trajectory, and each is evaluated using the cost function $\mathcal{J}$. The cost function $\mathcal{J}$ is defined as:

$$\mathcal{J} = \sum_i \alpha_i \mathcal{U}_i + \sum_i \beta i \mathcal{T}_i, \tag{2.5}$$

where $\mathcal{U}$ indicates the cost of the uncertainty, $\mathcal{T}$ represents the cost of the travel-to-goal, and α and β are weight parameters for these respective costs. The goal of active

SLAM is to select the policy that optimizes $\mathcal{J}$.

The approach proposed in [10] uses the particle filter to reduce the overall uncertainty for both maps and poses by capturing a trajectory's uncertainty using the particle weight. The Expectation-Maximization (EM) Exploration algorithm [11], [12] introduces *virtual landmarks* to represent the uncertainty of unexplored regions, and a novel utility function for solving the exploration problem that seeks the most accurate map possible with every sensing action.

2.2 Deep Reinforcement Learning

2.2.1 Deep Learning

Deep Learning [13] has achieved great success in many domains such as Computer Vision (CV), Natural Language Processing (NLP), and robotics. Convolutional Neural Networks (CNNs) [14], [15] are designed to analyze array-like data such as 2D images and 3D videos. The convolutional layer in CNNs is used for extracting local conjunctions of features from the previous output. The pooling layer is for combining semantically similar features. Recurrent Neural Networks (RNNs) are capable of processing sequential input data by memorizing the elements of the sequence in the hidden units. Typically, the long short-term memory (LSTM) architecture [16], [17] solves the vanishing gradient problem [18] in the traditional RNN model, because the cell and the gates in the LSTM model regulate the history information.

Analyzing graphs with deep learning is a fast-growing research area. Graph Neural Networks (GNNs) [19] use neural network models to process data in the graph domain. GNNs use the message passing function to capture the relationship between nodes in graphs and update the node or edge information during the data processing step. Graph Convolutional Networks (GCNs) [20] have a similar convolutional

mechanism as CNNs. Gate Recurrent Units (GRU) are proposed in the Gated Graph Neural Network (GGNN) [21] to unroll the recurrence steps and compute gradients by backpropagation through time. Graph Attention Networks (GAT) [22] adopt the attention mechanism for capturing important subsets of graphs.

2.2.2 Reinforcement Learning

Reinforcement Learning (RL) is capable of solving sequential decision-making problems by training an agent to predict the optimal action from the rewards collected in the environment. A Markov decision process (MDP) is the standard model used for reinforcement learning. The policy π guides the robot to choose an action corresponding to the current state. The policy expects to achieve the maximum discounted sum of future rewards R_t.

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.6}$$

Value-based methods [23], [24], [25] aim to maximize the expected sum of the rewards. The value function of a state-action pair is:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\{R_t | s_t = s, a_t = a\}. \tag{2.7}$$

Policy-based methods [26], [27], [28], [29] optimize the policy directly through gradient algorithms during the training process. The most common policy gradient form is:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi}[Q^{\pi}(s, a) \nabla_{\theta} \ln \pi_{\theta}(a|s)]. \tag{2.8}$$

2.3 Autonomous Exploration

2.3.1 Classical Exploration

Frontier-based exploration methods use the frontiers [30] which are boundaries between free and unknown areas. By taking a next-view point from frontiers in each decision-making time, the robot is guaranteed to explore the whole unknown environment eventually. However, the frontier-based exploration methods are not efficient in a 3D environment.

Information-theoretic exploration methods guide a robot to explore the unknown environment by repeatedly selecting the sensing action expected to be most informative [31], [32], [33]. Shannon's entropy [34] is typically employed as the information metric to describe the completeness of our knowledge about the contents of an occupancy grid map [35] m as follows:

$$H(m) = -\sum_{i}\sum_{j} p(m_{i,j}) \log p(m_{i,j}). \tag{2.9}$$

In Eq. (2.9), index i represents the individual grid cells of the map, and the possible outcomes of the Bernoulli random variable describing each map cell are referred to as index j. The Mutual Information (MI) $I(m, a_i)$ is used to evaluate the expected information gain with respect to the action a_t:

$$I(m, a_i) = H(m) - H(m|a_t), \tag{2.10}$$

An information-theoretic exploration method seeks to find an optimal sensing action at each decision-making step to obtain the maximum mutual information over an occupancy grid map. A fully explored environment can be achieved eventually,

which provides the minimum map entropy.

It has also proven effective to use Gaussian process frontier maps as a predictive tool to support exploration [36], [37]. However, these methods do not consider a robot's localization uncertainty during the exploration process, which eventually leads to an inaccurate map.

2.3.2 Learning-based Exploration

Learning-based exploration methods can offer reduced computation time and near-optimal exploration strategies. Bai et al. proposed a Gaussian process based mutual information prediction method [38], and a subsequent Bayesian optimization active sampling approach [39], to speed up the prediction of mutual information throughout a robot's action space. Supervised learning approaches have used labeled data collected from high-performance exploration methods to train neural networks to predict optimal exploration actions. Specifically, [40] shows that deep supervised learning can produce high-quality solutions for the exploration problem by using local occupancy maps as input data. Moreover, an exploration policy can be trained via reinforcement learning methods using representative example environments and a proper reward design. [1] and [41] demonstrate that a mobile robot can take occupancy grid maps as input data to predict the best exploratory action using reinforcement learning approaches. However, an occupancy grid map has limitations as an input to represent the environment. For example, in some instances, neural networks may be unable to perform reliably over a map with a rotation relative to the maps presented in training. Hence, such a framework may take a longer time to converge during the training phase, or even fail to converge, due to the size of the state space induced by an occupancy map.

Chapter 3

Learning-based Exploration without Uncertainty

This chapter will examine the potential of deep reinforcement learning (DRL) to match the performance of information-theoretic exploration. Combining DRL with local occupancy grid maps, a mobile robot can learn how to explore maze-like environments via the reward provided by the information gain.

3.1 Related Work

Model-based methods in RL often accelerate the training process [42], since the agent can obtain training information from a model in addition to the rewards from the environment. A pre-trained policy can also improve the performance of learning and increase a robot's learning efficiency [43]. Jaderberg proposed an auxiliary approach to explore the potentially reward-rich areas of an environment, avoiding low-reward areas [44]. Such methods require *a priori* knowledge of either the environments or the policy model.

Tai introduced a deep learning [45] and a deep reinforcement learning [46] based obstacle avoidance policy that is trained and tested using sensor data from the same environment. The learned obstacle avoidance policy can only be used in the same environment in which it is trained, and is ineffective for use across heterogeneous environments. Deep Recurrent Q-Networks (DRQN) were developed in [47] to play First-Person-Shooting (FPS) games in a 3D environment. By combining an RNN with a DQN, the DRQN can generate appropriate outputs that depend on the temporally consecutive inputs. Meanwhile, a target-driven robot visual navigation policy was also trained using deep reinforcement learning [48]. The robot was trained

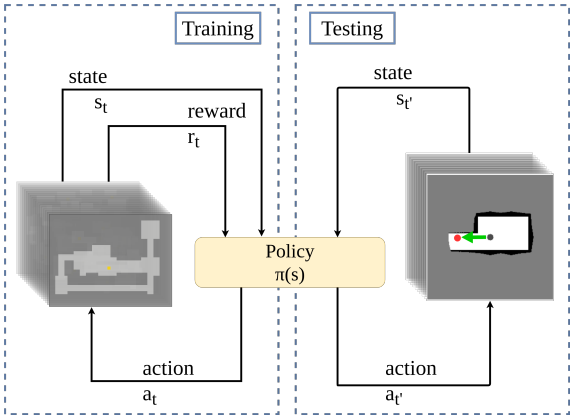


Figure 3.1: An illustration of our framework. In the *training phase*, we expose the robot to different environments and the policy optimizes parameters in the neural network by informative reward r_t, where the current state is a local map s_t and the corresponding robot action is a_t. Training concludes after 2 million episodes. Then in the *testing phase*, we provide a different set of diverse environments. The policy is used to estimate the optimal action, which is indicated by the red point a'_t from the current robot's local map s'_t.

in a simulated environment and implemented its policy successfully in a physical environment. However, this method requires a sophisticated simulation environment and it is challenging to predict the information gain of many potential future sensing actions using a single camera view. Choudhury et al. proposed the ExpLOre algorithm [49] to gather information using imitation learning. This algorithm provides non-myopic solutions for the autonomous exploration problem, however, the policy is trained and tested over the same maps, with different view nodes. Other works involving reinforcement learning and robot navigation in [50] and [51] have dealt with learning the topology of an environment as well as motion planning, however our work focuses purely on the efficiency of mapping in the absence of a prior map.

3.2 Problem Definition

Instead of choosing the sensing action of maximum expected information gain after a computationally expensive evaluation of all the candidate sensing actions, we employ

a neural network to predict the optimal sensing action and train it using reinforcement learning. We use a database of maps with similar characteristics, however with different individual topologies and layouts. A flowchart demonstrating both the training and testing phases is shown in Figure 3.1. We train and test the neural network on locally visible portions of randomly-generated 2D maps of indoor environments. We consider these locally visible, uniformly-sized regions instead of the entirety of a robot's currently acquired map with the aim of achieving a scalable, computationally efficient approach that may be applied to environments of arbitrary size. However, this is done with the understanding that we are training an information-seeking controller rather than a global optimization method.

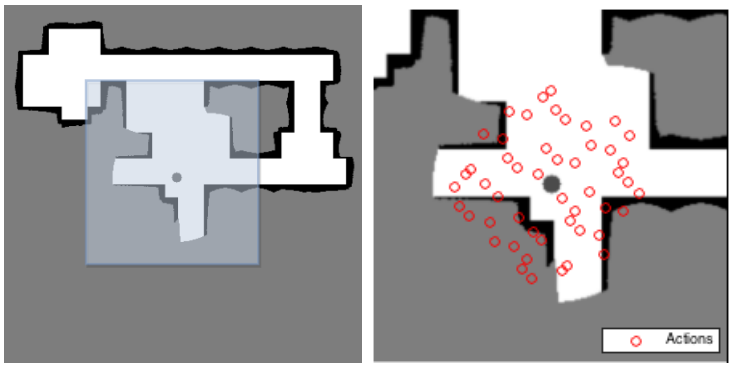
3.2.1 Entropy and Mutual Information

We use Shannon's entropy as the metric to represent the completeness of our knowledge about the map, which is described in the following equation:

$$H(m) = -\sum_{i}\sum_{j} p(m_{i,j}) \log p(m_{i,j}). \tag{3.1}$$

In Eq. (3.1), m represents the current occupancy grid map, where index i refers to the individual grid cells of the map and index j refers to the possible outcomes of the Bernoulli random variable that represents each grid cell, which is either free or occupied. Cells whose contents have never been observed are characterized as $p(m_{i,j}) = 0.5$, contributing one unit of entropy per cell. Cells whose contents are perfectly known contribute no entropy to the summation.

We define Mutual Information (MI) $I(m, a_i)$ to be the expected information



(a) (b)

Figure 3.2: Left: A local map is extracted from the global map and is used to represent the state space. The gray point in the free space indicates the location of the robot itself. Right: An illustration of the same robot's action space.

gain with respect to the action a_t:

$$I(m, a_i) = H(m) - H(m|a_t), \tag{3.2}$$

where $H(m)$ is the entropy of the current map, and $H(m|a_t)$ is the expected entropy after collecting a sensor observation upon executing the action a_t.

3.3 Deep Reinforcement Learning

3.3.1 State Space and Action Space

We initially adopted the robot's entire global map as the state space S_{global}, however it was challenging to achieve efficient convergence for such a large state space, and map size may vary widely from one application to the next. Thus, we instead choose a local region of the map, centered at the robot's current location, as our state space S as shown in Figure 3.2(a). We define the action space A as a set of quasi-random samples drawn within a constant radius of the robot, shown in Figure 3.2(b) (here we use the Sobol sequence).

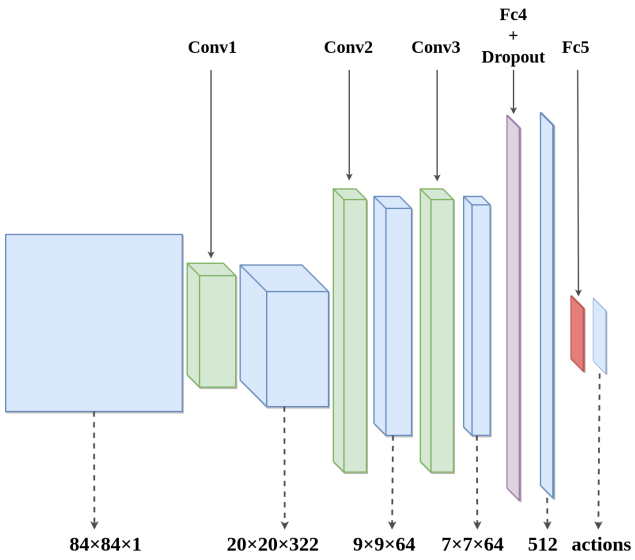


Figure 3.3: An illustration of our convolutional neural network, in which the blue cubes represent each data structure. The original input data is a single-frame grayscale image. The output of the network is a one-dimensional vector, where each entry is the reward associated with one specific action.

3.3.2 Reward Design

Mnih et al. [24] has suggested normalizing the reward so that $r \in [-1, 1]$. We choose 70 percent of the maximum possible mutual information from Eq. 3.2 as the upper bound for the reward (which is rarely exceeded in practice). We also normalize the MI acquired by an action a_t using 70 percent of the maximum possible MI, if the MI obtained is greater than zero. When an action leads to collision with an obstacle or zero information gain, we assign $r = -1$ and $r = -0.8$ repectively. The reason for assigning negative reward to actions with zero information gain is to discourage driving to previously observed areas of the map.

3.3.3 Value Function

The function V^{π} is the state-value function for policy π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\{R_t | s_t = s\}. \tag{3.3}$$

In (3.3), $\mathbb{E}_{\pi}$ indicates the expected discounted future reward R provided by the policy π in terms of the current state at time-step t. Furthermore, the function Q^{π} is the state-action value function for policy π:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\{R_t | s_t = s, a_t = a\}. \tag{3.4}$$

In (3.4), at time-step t, not only the state but also the action are utilized to generate the expected discounted future reward R provided by the policy π. Both value functions V^{π} and Q^{π} are used to evaluate the current state and action. Using these value functions, the agent can identify whether the current state and action yield a high reward. Specifically, the larger number shows the better circumstance.

3.3.4 Reinforcement Learning Framework

The typical setting for reinforcement learning (RL) [52] is a partially observable Markov decision process (POMDP). The agent chooses an action guided by policy π based on current state S, and expects to maximize its discounted future reward R, which is represented as follows:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{3.5}$$

where $\gamma \in [0, 1]$ denotes the discount rate. By setting a higher value for γ, it will encourage the model to pay more attention to future rewards. In contrast, the model's training will be more focused on the current action if γ decreases.

We use a Deep Reinforcement Learning (DRL) model and define the transition tuple as $< s, a, r, s' >$, where s denotes a robot's current state, a denotes its action, r denotes the reward and s' denotes the next state, achieved by the transition. The Bellman optimality equation for $Q^{\pi}(s, a)$ is:

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}\{r_t + \gamma \max_{a' \in A} Q^*(s_{t+1}, a') | s_t = s, a_t = a\} \\ &= \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_{a' \in A} Q^*(s', a')] \\ &= r + \gamma \max_{a' \in A} Q^*(s', a'), \end{aligned} \tag{3.6}$$

where $\mathcal{P}_{ss'}$ are the transition probabilities and $\mathcal{R}^a_{ss'}$ is the expected reward for the current transition. In our model, the transition probabiliy from s_t to s_{t+1} is 1, and thus $s' = s_{t+1}$ and $\mathcal{P}^a_{ss'} = 1$. The policy is defined as follows:

$$\pi(s) = \arg\max_{a \in A} Q(s, a), \tag{3.7}$$

and the loss function is defined as follows:

$$L = \frac{1}{2}[r + \gamma \max_{a' \in A} Q^*(s', a') - Q^*(s, a)]^2. \tag{3.8}$$

The loss of the action-value function is the squared difference of the policy-provided value and the Bellman optimality equation-provided value. We want to minimize the loss through training, so that the policy π can select a near-optimal action even if it misses the optimal one. We use the Adam first-order gradient-based

algorithm [53] when optimizing the parameters of the DRL model.

3.3.5 Neural Networks

We integrated two different neural networks in our framework, including the convolutional neural network (CNN) used for Q-learning in [24], as shown in Fig. 3.3, and a Long Short Term Memory network (LSTM) [54], which is a state-of-the-art structure for RNNs. The output is influenced by the temporal sequential inputs to the RNN model, offering the prospects of decision making that considers the temporal context of an input state. Furthermore, we have added a dropout layer to both neural networks to avoid overfitting.

3.3.6 Training Strategies: Bayesian

The work of [55] has proven that the output of the dropout [56] neural network layer can represent the uncertainty of the actions. With probability p, the output of the dropout layer is the input element scaled up by $1/p$. For the networks with a dropout layer, we first define $p = 0.1$ to choose the most uncertain action, and as the training phase runs, p is increased gradually until $p = 1$, which means the robot always take an action using the policy.

3.4 Experiments and Results

3.4.1 Experimental Setup

We first trained a policy over a 2D maze-like environment, using 5663 maps during the training phase. The simulated robot in our experiment is provided a 360-degree field of view noise-free range sensor. Its occupancy grid map is handled deterministically, where $p = 1$ for occupied cells, $p = 0$ for free cells and $p = 0.5$ for unknown cells.

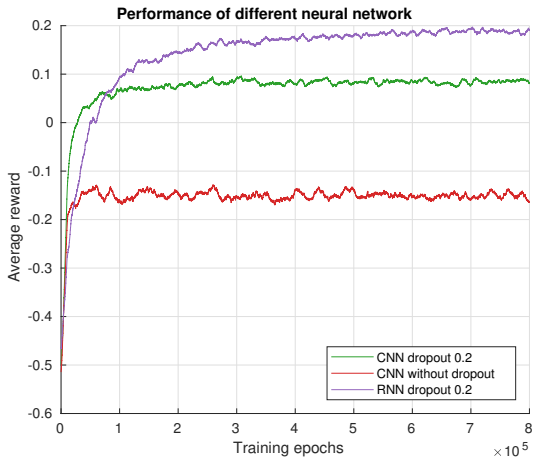
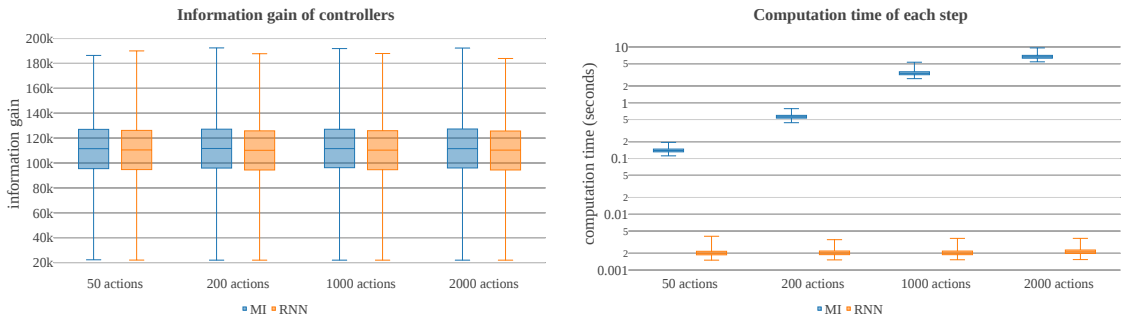


Figure 3.4: Convergence rate of neural networks during training. Both the CNN and RNN benefit from having a dropout layer, which improved not only the convergence rate but also the average reward.

Every map is $640m \times 480m$, while the sensor range and each robot step are 80m and 40m, respectively. We use a $240m \times 240m$ local submap captured at the robot's current location as the input state to the neural network. We translate the local map to a grayscale image, where 255 indicates free space, 1 indicates occupied space, and 127 is unexplored space. Further, we denote the robot in the map as a circle with a 6m radius, and it is assigned a grayscale value of 76.

The robot checks for collisions before taking any action, and will include instances of collision in its training. However, the robot will only proceed with exploring after a collision-free action is selected. If the robot navigates into a dead end, this will trigger a "frontier rescue (FR)" strategy, which drives the robot to the nearest frontier [30] in its occupancy map, so that we may continue training with the current map. We explored different neural networks for training, as described in Section 3.3.5.

The simulation environment is written in Python, with a C++ library for the inverse sensor model, and our neural network model is trained using TensorFlow [57]. However, the inverse sensor model cannot be processed in parallel because



(a) (b)

Figure 3.5: (a): A comparison of information gain achieved by the MI-maximizing controller and the RNN approaches, over action spaces of different sizes. (b): The average computation time for each decision making step, over action spaces of different sizes. Note that both plots show not only the mean and standard deviation (wide bars), but also the minimum and maximum value of information gain and computation time (narrow bars) among all trials.

the workload of each ray casting operation is too low compared with the associated overhead. The training and testing maps are generated randomly by [58].

3.4.2 Testing in a 2D environment

We compared the learned RNN controller and an exhaustive MI-optimizing controller on a testing-phase dataset comprised of 5218 maps. The MI controller, along the lines of [33], explicitly evaluates the expected information gain of every possible sensing action, by projecting the sensor's rays throughout the map at each candidate configuration, and choosing the sensing action that offers maximum expected MI. A frontier based rescue mechanism will take place when there is zero information gain resulting from any action. For both methods, the exploration of a map will be terminated only after the entropy of a map has been reduced to zero. The testing of the mutual information controller was performed using an Intel i7 3.5Ghz CPU, while the learned RNN controller was run on an Nvidia GeForce GTX 1080 GPU.

The FR heuristic is introduced to lead the robot to the nearest frontier from

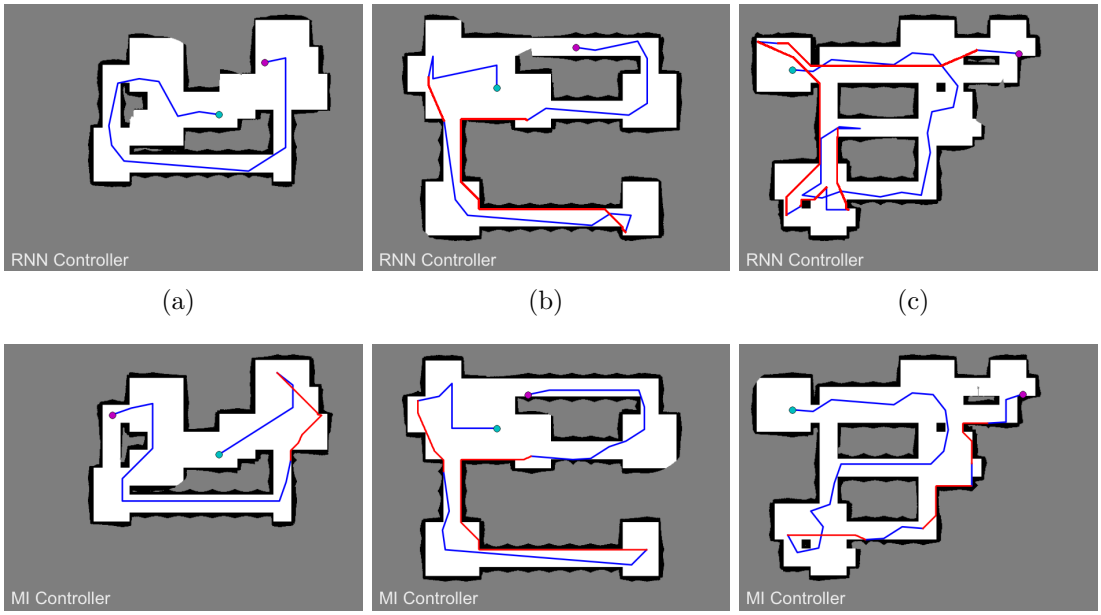


Figure 3.6: Representative trajectories produced by the RNN controller (top) and the exhaustive MI controller approach (bottom), over three example maps from our testing phase dataset. The robot starts from the cyan-colored point and ends at the magenta-colored point in each trial (at which point the map has been fully explored). When the robot is stuck in a local mapped region, the frontier rescue method will lead the robot to go to the nearest frontier point by using the A* path planning algorithm, which is indicated by the red lines above. The blue lines are the trajectories provided by the RNN controller and the MI controller. Subfigures (a) and (d) show an RNN controller instance that outperforms the MI controller, (b) and (e) show similar outcomes among the two methods, and (c) and (f) illustrate a lower-performing RNN controller instance relative to the MI controller.

obstacle-induced or information-poor dead-ends, where the A* algorithm finds an efficient path to the nearest frontier from the robot's current location. We believe employing an efficient planning algorithm such as A* is a reasonable approach for navigating from *known* areas of the map back to the frontiers, as the focus of this study is learning to select informative sensing actions in partially unknown environments, rather than navigating known maps.

We select two metrics to show the relative performance of both methods. The

total information gain from controller-driven sensing actions is a key element to evaluating the performance of the methods, while the information gain acquired by FR heuristic has been excluded. We assume the robot executes each sensing action at the same, constant velocity, as it is straightforward that the computation time of each decision making step is also a critical index to compare the efficiency of the two controllers.

Fig. 3.4 demonstrates the outcomes of the training phase for the case of 50 actions, giving a comparison among all of the neural networks examined. In two cases, we set a dropout layer with a rate of 0.2 in the penultimate layer of the neural network. The dropout strategy improves the final converged average reward for both the RNN and CNN. The CNN without a dropout layer offers the worst reward result in the end. Meanwhile, the CNN with a 20 percent dropout layer achieves a higher average reward. Although the RNN has a slower rate of convergence than the other networks examined, because it needs to learn sequential map information, it provides the best average reward in the end.

Fig. 3.5 gives results showing the performance of the exploration methods over different-sized action spaces in the testing phase. We use box plots to describe the distribution of the testing set of data. Each box plot represent 5218 data which correspond to the number of testing maps. The x-axis of each plot indicates the size of the action space examined. MI indicates a traditional exhaustive MI-maximizing controller. The RNN approach indicates the learned RNN controller from the training phase. In Fig. 3.5(a), the y-axis indicates the total amount of information gain obtained only by the controllers in each map. The RNN controller offers nearly the same performance as the traditional MI approach across the different action spaces. In Fig. 3.5(b), the log-scale y-axis indicates the computation time required for decision making at each step. The computation time of the exhaustive MI-maximizing

approach grows sharply as the action space grows. Meanwhile, the computation time of the RNN method remains constant as the action space grows.

Fig. 3.6 shows a comparison of representative trajectories generated by the MI-maximizing controller and our RNN controller over the same three maps. Figs. 3.6 (a), (b) and (c) are produced by the learned RNN controller, and (d), (e) and (f) are generated by the MI-maximizing controller. Trajectory (a) offers a non-cyclical trajectory which explores more efficiently than the outcome of (d). Trajectories (b) and (e) show a similar result produced by the two respective methods. Trajectory (c) has a worse exploration outcome because FR is implemented several more times than in (f). However, in spite of the occasional need for frontier-guided paths, the trajectory portions produced by the learned RNN controller are straightforward and efficient.

Finally, a supplementary video offers three examples of how our RNN controller performs in the testing phase, after different quantities of training epochs[1]. As the learned policies converge over the course of two million epochs, each of the three examples manages to explore its map in entirety with an exclusive reliance on the RNN controller, without the need for the frontier rescue heuristic.

3.4.3 3D exploration

Furthermore, we validate the proposed approach in a 3D synthetic environment. Due to the complexities of training over a much denser input, in this experiment we train and test on the same map, but from different start locations. The testing of the MI controller was performed using an Intel i7 4.2Ghz CPU, while the learned RNN controller was run on an Nvidia Geforce Titan X Pascal GPU. We equipped the robot with a simulated Velodyne VLP-16 3D LiDAR Sensor which provides a 360-

[1] `https://youtu.be/2gNF6efv12s`

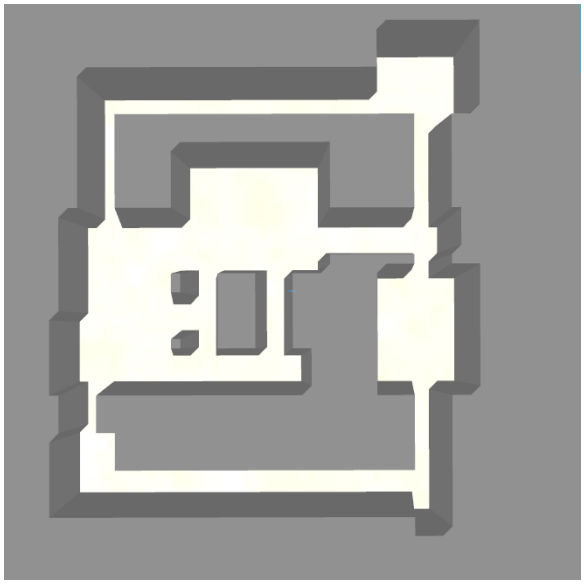


Figure 3.7: A 3D synthetic environment (top view) used for both training and testing.

degree horizontal field of view and a 30-degree vertical field of view. Fig. 3.7 shows the environment used for 3D exploration. The robot was initialized from random locations in the map during training, and a selected set of locations when testing (close to the corners of the map). Fig.3.8 shows an example sensing action selected by the RNN controller. We use a similar RNN framework (which takes a more dense input) and trained it from scratch. We use a $30m \times 30m \times 0.8m$ local occupancy grid map with $0.2m$ resolution as the input state. We assigned a value 10 to indicate occupied cells, 255 for free cells and 127 for unknown cells. The action space is a set of 200 quasi-random points within a 5m radius in the horizontal plane. We apply the same strategy when developing the reward function.

In Fig. 3.9, the information gain received only by the MI baseline approach and the RNN approach, without the involvement of the FR heuristic, is shown for our 3D scenario. The robot will stop exploration if it maps 2500 m^3 of the free volume of the map (the total volume of the free space is 3200 m^3). The y-axis of the plot shows the average free volume mapped from four different starting points at the corners of

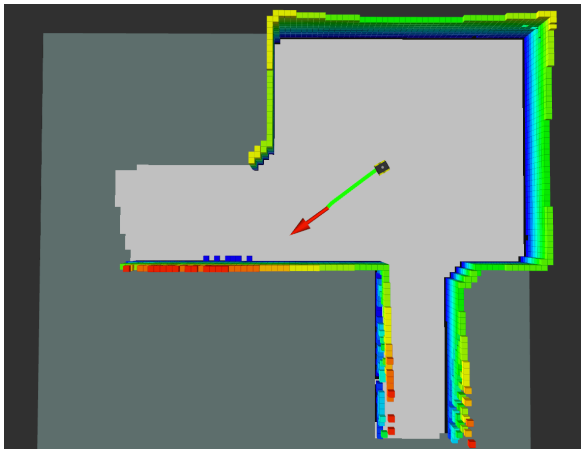


Figure 3.8: The RNN controller uses a local 3D occupancy grid submap as the input data and estimates the best sensing action for the robot.

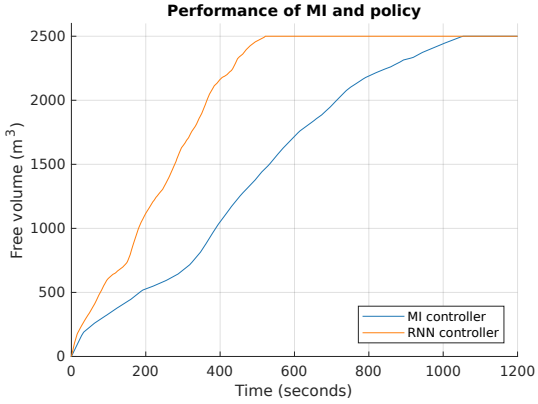


Figure 3.9: Comparison of information gain between the RNN and MI controllers in a 3D synthetic environment (average of 6 trials). Note there is no frontier rescue here, and the robot will explore until the mapped free volume reaches the designated threshold, 2500 m^3.

the map. The result shows that the RNN policy leads to a 100 percent increase in efficiency in reaching the free-volume goal, compared with the MI method.

In addition to showing three examples of how our RNN controller performs in the testing phase over 2D maps, the video attachment also shows the full 3D exploration process for both the MI and RNN controllers. The robot stops when it has mapped a designated fraction of the available free volume of the map (90 percent of the total free volume). As depicted in Fig. 8, the timing of 3D exploration shown in the video also includes the computation time associated with robot decision making

under each of the competing autonomous exploration frameworks.

3.5 Conclusions

We have proposed a novel approach to estimate the best mobile robot sensing action in the course of exploring an unknown environment, trained via deep reinforcement learning. In our framework, the DRL policy generates robot sensing actions that are nearly as informative and efficient as those of a standard mutual information maximizing controller, at a substantial reduction in computational effort during the online testing phase. The time complexity in the testing phase is O(1), which means that our proposed method is scalable to higher-dimensional state-action spaces, as long as the policy has been trained in a space of the same dimension. For the mutual information optimizing controller, with increasing dimension of the configuration space and action space, the procedure's computational complexity is no longer be real-time viable. Looking ahead to future work, we also note that it is difficult to identify useful and descriptive frontiers in a 3D map.

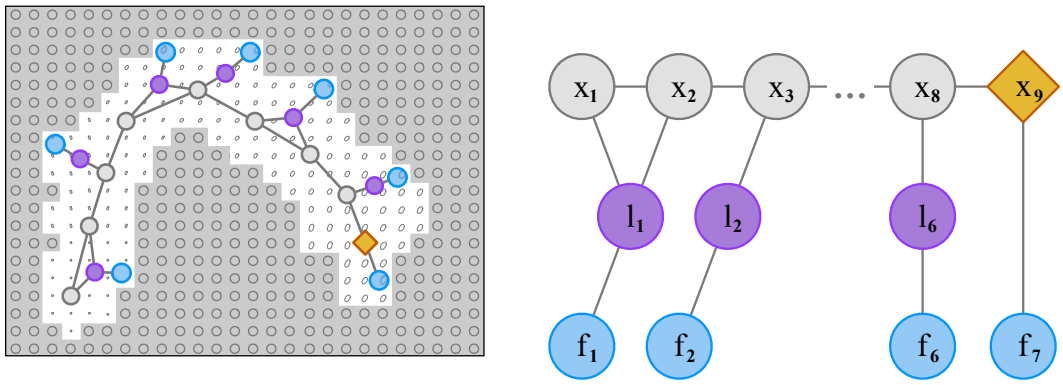
Chapter 4
Learning-based Exploration under Uncertainty

In this chapter, we will consider a learning-based approach to mobile robot exploration of unknown environments that captures localization uncertainty. The robot learns how to explore an unknown environment with the highest exploration efficiency as well as the lowest map error. The proposed graph-based input data can capture the topological structure and the key information of the current environment. The learned policy can be used in a dynamic state-action space environment.

4.1 Related Work

Standard exploration algorithms use forward simulation to evaluate the mapping outcomes and uncertainty of next-view candidates, and so their computational cost increases substantially with an increasing number of next-view candidates. Learning-based exploration methods offer the prospect of improved scalability for selecting the next best view, with and without localization uncertainty. Learning-aided information-theoretic approaches [38], [39] are proposed to reduce the cost of predicting MI. Deep neural networks in [40], [1] and [41] can be trained to predict the optimal next-view candidate through supervised learning and reinforcement learning, by taking occupancy grid maps as input data. However, the state space of such maps is very large, and a learned policy may fail when tested in a completely new environment.

Graphs can offer generalized topological representations of robot navigation, permitting a much smaller state space than metric maps. GNNs incorporate graphs into neural network models, permitting a variety of queries to be posed over them [19]. Sanchez-Gonzalez et al. [59] proposed to use graph models of dynamical sys-



(a) (b)

Figure 4.1: **Formulating and extracting the exploration graph.** Left: the grayscale color represents the probability of occupancy of a given map cell (assumed to be 0.5 for unobserved cells). The ellipse in each cell represents the estimation error covariance of each *virtual landmark*, with true landmarks shown in purple. Past robot poses, the current pose, and candidate frontiers are indicated by gray circles, an orange diamond, and blue circles respectively. Landmark nodes and the current pose node are connected with their nearest frontiers. Right: An input exploration graph is extracted from the current exploration state. Each edge in this graph is weighted with the Euclidean distance between the two vertices connected.

tems to solve control problems with Graph Nets [60]. Exploration under localization uncertainty is achieved in our prior work [2] through supervised learning over graphs, which gives an unstable result because the number of nodes in each graph differs, and the hyperparameters of the loss function are hard to tune. A novel graph localization network is proposed in [61] to guide a robot through visual navigation tasks. Wang et al. [62] introduces GNNs as policy networks and value networks, through their integration into DRL, to solve control problems.

4.2 Problem Formulation and Approach

4.2.1 Simultaneous Localization and Mapping Framework

We adopt a graph-based approach in the SLAM framework supporting our robot's exploration. We then solve the SLAM problem as a least-squares smoothing problem. The robot motion model and measurement models are defined as:

$$\mathbf{x}_i = h_i(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}_i, \quad \mathbf{w}_i \sim \mathcal{N}(\mathbf{0}, Q_i), \tag{4.1}$$

$$\mathbf{z}_k = g_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, R_k), \tag{4.2}$$

where $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{t}$ are poses along the trajectory, $\mathcal{L} = \{\mathbf{l}_j\}_{j=1}^{m}$ are $m \in \mathbb{N}$ landmarks, and $\mathcal{U} = \{\mathbf{u_i}\}_{i=1}^{t}$ is a given sequence of low-level actions. Then we solve the SLAM problem as a least-squares problem:

$$\begin{aligned} \mathcal{X}^*, \mathcal{L}^* = \arg\min_{\mathcal{X},\mathcal{L}} & \sum_i \left\| \mathbf{x_i} - h_i(\mathbf{x_{i-1}}, \mathbf{u_i}) \right\|_{Q_i}^2 \\ & + \sum_k \left\| \mathbf{z_k} - g_k(\mathbf{x_{i_k}}, \mathbf{l_{j_k}}) \right\|_{R_k}^2 . \end{aligned} \tag{4.3}$$

$\mathcal{X}^*$ and $\mathcal{L}^*$ are obtained using the GTSAM [63] implementation of iSAM2 [64] to perform nonlinear least-squares smoothing over a factor graph. The Gaussian marginal distributions and Gaussian joint marginal distributions are produced from this graph-based inference procedure.

We next introduce the *virtual map* of the EM algorithm for exploration under localization uncertainty [11], which is a uniformly discretized grid map comprised of *virtual landmarks*, $\tilde{\mathbf{l}}_k \in \tilde{\mathcal{L}}$, to represent the uncertainty and occupancy probability of every map cell. Each cell of the virtual map contains a virtual landmark with a large initial covariance (illustrated in Figures 4.1(a) and 4.2(a)). We use A-optimality [65]

for our uncertainty criterion:

$$\phi_{\mathrm{A}}(\Sigma) = \mathrm{tr}(\Sigma), \tag{4.4}$$

where Σ is the error covariance matrix for all virtual landmarks in the map. Accordingly, the uncertainty of the current state is quantified by the trace of the covariance of all virtual landmarks. The definition of the utility function is as follows:

$$U(\tilde{\mathcal{L}}) = \sum_{\tilde{\mathbf{l}}_k \in \tilde{\mathcal{L}}} \phi_{\mathrm{A}}(\Sigma_{\tilde{\mathbf{l}}_k}). \tag{4.5}$$

4.2.2 Exploration Graph

The definition of the *exploration graph* is $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{X} \cup \mathcal{L} \cup \mathcal{F}$, and where $\mathcal{X}, \mathcal{L}$, and $\mathcal{F}$ are sets of previous poses, landmarks, and candidate frontier nodes respectively. The edges $\mathcal{E}$ connecting pose to pose $\mathbf{x}_i$ — $\mathbf{x}_{i+1}$, pose to landmark $\mathbf{x}_{i_k}$ — $\mathbf{l}_{j_k}$, landmark to frontier $\mathbf{l}_{j_k}$— $\mathbf{f}_{n_k}$, and the current pose to its nearest frontier $\mathbf{x}_t$— $\mathbf{f}_{n_t}$ are weighted with the Euclidean distances between those nodes. Each node $\mathbf{n_i} \in \mathcal{V}$ has a feature vector:

$$\mathbf{s}_i = [s_{i_1}, s_{i_2}, s_{i_3}, s_{i_4}, s_{i_5}],$$

$$s_{i_1} = \phi_{\mathrm{A}}(\Sigma_i), \tag{4.6}$$

$$s_{i_2} = \sqrt{(x_i - x_t)^2 + (y_i - y_t)^2}, \tag{4.7}$$

$$s_{i_3} = \mathrm{arctan2}(y_i - y_t, x_i - x_t), \tag{4.8}$$

$$s_{i_4} = p(m_i = 1), \tag{4.9}$$

$$s_{i_5} = \begin{cases} 0 & \mathbf{n_i} = \mathbf{x_t} \\ 1 & \mathbf{n_i} \in \{\mathbf{f_n}\} \\ -1 & \text{otherwise} \end{cases}. \tag{4.10}$$

The contents of the feature vector are as follows. In Eq. (4.6), the A-Optimality criterion, derived from our virtual map $\tilde{\mathcal{L}}$, is used to quantify a node's uncertainty. Eqs. (4.7) and (4.8) provide relative pose information; the Euclidean distance and the relative orientation between the current robot pose and each node in the exploration graph. Occupancy information is extracted from an occupancy grid map, where m_i is the occupancy of the map cell associated with node n_i in Eq. (4.9). Eq. (4.10) provides an indicator variable, denoting the current pose to be 0, all frontiers to be 1, and all other nodes -1.

We note that frontier nodes are sampled from the boundary cells between free and unexplored space in the occupancy map. The landmarks and the current robot pose are the only nodes connected to frontiers, and each connects only to its nearest frontier node. It is possible for multiple landmark nodes to be connected to the same frontier node, but not for a landmark or pose node to connect to multiple frontier nodes; only the nearest frontiers are connected into the graph. The composition of an exploration graph is shown in Fig. 4.1.

4.3 Supervised Learning

Instead of explicitly computing the sensing action that maximizes the reward through an expensive evaluation of all the candidate frontiers, we employ a GCN to predict the optimal frontier with respect to the utility gain. However, we use the EM algorithm to generate training data. For each decision-making step, actions $\mathcal{A} = \{\mathbf{a}_n\}$ are generated for each frontier node with straight-line path planning, and the EM algorithm provides an action reward $\mathcal{R} = \{r_n\}$ for all frontiers via forward simulation. The reward $\mathcal{R}$ is normalized so that $r_n \in [0, 1]$. We require that the reward of an optimal frontier has to be larger than 0.95. If multiple frontiers are optimal

at the same time, we will select one at random. The rewards associated with pose and landmark nodes are always 0 because we only select the robot's next action from among frontier nodes. Therefore, the training label y_{nodes} is defined as follows:

$$y_i = \begin{cases} 1 & r_i > 0.95 \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

A multi-layer GCN is adopted to explore the environment. The layer-wise propagation rule of the GCN [20] is as follows:

$$H^{(l+1)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}), \tag{4.12}$$

with $\hat{A} = A + I$, where A is the adjacency matrix of the exploration graph $\mathcal{G}$ and $\hat{D}$ is the degree matrix of $\hat{A}$. $H^{(l)}$ represents the lth hidden layer of the GCN model with the activation function $\sigma(\cdot)$. $W^{(l)}$ is the weight matrix of the lth layer.

For the GCN model, the definition of each hidden layer is as follows:

$$H^{(1)} = \sigma_1(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}SW^{(1)}), \tag{4.13}$$

$$H^{(2)} = \text{Dropout}(H^{(1)}), \tag{4.14}$$

$$H^{(3)} = \sigma_3(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(2)}W^{(3)}). \tag{4.15}$$

In Eq. (4.13), the size of weight matrix $W^{(1)}$ is 5×1000 because we want to upsample the number of features from 5 to 1000. The activation function σ_1 is a Rectified Linear Unit (ReLU). We add a dropout layer with 0.5 dropout rate as shown in Eq. (4.14). In the output layer, the size of the weight matrix $W^{(3)}$ in Eq. (4.15) is 1000×1 and the activation fuction σ_3 is a sigmoid function.

The cross-entropy loss function is defined as follows:

$$L = -(w_1 y \log\hat{y} + w_0(1-y)\log(1-\hat{y})), \tag{4.16}$$

where w_1 and w_0 are the weights for class 1 and class 0. These weights are used to balance the cost for imbalanced data. We empirically define $w_1 = 21$ and $w_0 = 1$. We use the Adam first-order gradient-based algorithm [53] when optimizing the parameters of the GCN model.

4.4 Deep Reinforcement Learning

Reinforcement learning describes a sequential decision making problem, whereby an agent learns to act optimally from rewards collected after taking actions. In our setting, we consider changes to the exploration graph that result from our selection of frontier nodes, which represent waypoints lying on the boundaries between mapped and unmapped areas. At each step $k \in \mathbb{N}$ the environment state is captured by the exploration graph $\mathcal{G}_k \in \mathsf{G}$ [1]. The robot chooses a frontier node to visit based on the graph topology: $\mathbf{f}_k \in \mathsf{F}_{\mathcal{G}_k}$. This causes a transition to $\mathcal{G}_{k+1} \in \mathsf{G}$ and a scalar reward $R \in \mathbb{R}$ to be emitted. The interaction is modeled as a Markov Decision Process $\langle \mathsf{G}, \mathsf{F}, \Pr, \gamma \rangle$ [66], associated with the transition kernel $\Pr\colon \mathsf{G} \times \mathsf{F} \to \mathscr{P}(\mathsf{G} \times \mathbb{R})$ that defines a joint distribution over the reward and next exploration graph, given the current graph and frontier node pair. Here, $\gamma \in [0, 1)$ is a discount factor used to control the relative importance of future rewards in the agent's learning objective.

In this chapter we consider value-based methods and policy-based methods for model-free control. Value methods strive to maximize the expected sum of future

[1] The exploration graph $\mathcal{G}$ encodes the full history of robot poses.

Algorithm 4.1: Reward Function

input: Exploration graph $\mathcal{G}$, Frontier node $\mathbf{f}$
Normalize the raw reward Alg. 4.2
$\mathcal{R}_\mathcal{G} = \{r_{\mathbf{f}'} = \texttt{raw\_reward}(\mathcal{G}, \mathbf{f}') \; \forall \; \mathbf{f}' \in \mathcal{A}_\mathcal{G}\}$
$l = \min \mathcal{R}_\mathcal{G}$, $u = \max \mathcal{R}_\mathcal{G}$
$r_\mathbf{f} \leftarrow (r_\mathbf{f} - l)/(u - l)$
Compute projection based on nearest frontier
$\mathbf{f}_t = \texttt{nearest\_frontier}(\mathbf{x}_t)$
if $u =$ *raw_reward*$(\mathbf{f}_t)$ **then**
    return $r_\mathbf{f} - 1$ # $r(\mathcal{G}, \mathbf{f}) \in [-1, 0]$
return $2r_\mathbf{f} - 1$ # $r(\mathcal{G}, \mathbf{f}) \in [-1, 1]$

Algorithm 4.2: Raw reward

input: Exploration graph $\mathcal{G}$, Frontier node $\mathbf{f}$
$\mathcal{U} \leftarrow \texttt{path\_planner}(\mathcal{G}, \mathbf{f})$
Compute raw reward with (4.5), Alg 4.3 and cost-to-go
return $U(\tilde{\mathcal{L}}) - \texttt{compute\_utility}_{\tilde{\mathcal{L}}}(\mathcal{U}) - \alpha C(\mathcal{U})$

rewards, or the *value*:

$$Q_{\boldsymbol{\theta}}(\mathcal{G}, \mathbf{f}) = \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R(\mathcal{G}_k, \mathbf{f}_k) \;\middle|\; \mathcal{G}_0 = \mathcal{G}, \mathbf{f}_0 = \mathbf{f}\right]. \tag{4.17}$$

For large-scale problems, the value function is represented with a parameterized function, such as a convolutional neural network or, in our study, a graph neural network.

To train $Q_{\boldsymbol{\theta}}$, we gather transition samples of the form $(\mathcal{G}, \mathbf{f}, r, \mathcal{G}')$ using an action sampling strategy that chooses the most uncertain action, which has the largest value at the output. According to [55], the uncertainty of the actions can be represented by the output of the dropout layer. During training, we first dropout 90% of our data before the MLP output model, and as the training phase runs, we gradually decrease the dropout rate until it reaches 0%, which means the policy provides greedy action selection.

Algorithm 4.3: Compute Utility

global: Virtual landmarks $\tilde{\mathcal{L}}$
input: Sequence of low-level actions $\mathcal{U}$
Execute low-level actions with [11] $\tilde{\mathcal{L}} \leftarrow$`update_virtual_map`$(\mathcal{U})$
Compute uncertainty estimate with (4.5)
return $U(\tilde{\mathcal{L}})$

The parameters $\boldsymbol{\theta}$ are adjusted using stochastic gradient descent to minimize the loss function $L\colon \mathcal{B} \to \mathbb{R}$ over sampled minibatches $\mathcal{B} \sim \mathcal{D} = \{(\mathcal{G}, \mathbf{f}, r, \mathcal{G}')_k\}_{k\in\mathbb{N}}$. In this chapter we consider the DQN [24] loss function:

$$L_{\mathrm{DQN}}(\mathcal{D}) = \mathbf{E}_{\mathcal{B}\sim\mathcal{D}}[(r + \gamma \max_{\mathbf{f}'\in\mathsf{F}_{\mathcal{G}'}} Q(\mathcal{G}', \mathbf{f}') - Q(\mathcal{G}, \mathbf{f}))^2], \tag{4.18}$$

which encodes the expected squared TD-error of one-step predictions over $\mathcal{B}$.

We also consider the policy-based method A2C [67], that directly trains a parameterized policy $\pi_{\boldsymbol{\theta}}\colon \mathsf{G} \to \mathscr{P}(\mathsf{F})$ from data gathered following that policy. We use two separate GNN models to serve as the policy network and the value network, and train it with the loss function:

$$\begin{aligned} L_{\mathrm{A2C}}(\mathcal{D}) &= \mathbf{E}_{\mathcal{B}\sim\mathcal{D}}[L^{(1)}_{\mathrm{A2C}} + \eta L^{(2)}_{\mathrm{A2C}}], \\ L^{(1)}_{\mathrm{A2C}} &= [A(\mathcal{G}, \mathbf{f}) - V(\mathcal{G}))\log\pi(\mathbf{f}|\mathcal{G}) + \beta(A(\mathcal{G}, \mathbf{f})]^2, \\ L^{(2)}_{\mathrm{A2C}} &= \sum_{\mathbf{f}\in\mathsf{F}_{\mathcal{G}}} \pi(\mathbf{f}|\mathcal{G})\log\pi(\mathbf{f}|\mathcal{G}). \end{aligned} \tag{4.19}$$

Here, $L^{(1)}_{\mathrm{A2C}}$ and $L^{(2)}_{\mathrm{A2C}}$ denote the loss terms for a single transition sample. The function $A(\mathcal{G}, \mathbf{f}) = Q(\mathcal{G}, \mathbf{f}) - V(\mathcal{G})$ is called the *advantage*; it computes the difference between the state-action value function Q and the state value function $V(\mathcal{G}) = \max_{\mathbf{f}\in\mathsf{F}_{\mathcal{G}}} Q(\mathcal{G}, \mathbf{f})$. We use $\beta \in \mathbb{R}$ as a coefficient for the value loss, and we use $\eta \in \mathbb{R}^+$ as a coefficient for the entropy of the output, to encourage exploration within the RL

Algorithm 4.4: Exploration Training with RL

initialize: $\mathcal{G}$, $\boldsymbol{\theta}$, *step*
$\mathcal{D} \leftarrow \emptyset$
while *step* ¡ *max_training_steps* **do**
    if *RL="DQN"* **then**
        # Gather experience visiting frontier nodes
        $\mathbf{f} \leftarrow \texttt{action\_sampling}(\mathcal{G})$
        $\mathcal{G}', r \leftarrow \texttt{visit\_frontier}(\mathcal{G}, \mathbf{f})$
        $step \leftarrow step + 1$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathcal{G}, \mathbf{f}, r, \mathcal{G}'\}$
        $\mathcal{G} \leftarrow \mathcal{G}'$
        # Train policy with DQN algorithm
        $\pi_{\boldsymbol{\theta}} \leftarrow \texttt{dqn}(\mathcal{D})$
    else if *RL="A2C"* **then**
        # Gather experience visiting frontier nodes
        $\mathbf{f} \leftarrow \pi_{\boldsymbol{\theta}}(\mathcal{G})$
        $\mathcal{G}', r \leftarrow \texttt{visit\_frontier}(\mathcal{G}, \mathbf{f})$
        $step \leftarrow step + 1$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathcal{G}, \mathbf{f}, r, \mathcal{G}'\}$
        $\mathcal{G} \leftarrow \mathcal{G}'$
        # Train policy with A2C algorithm
        if $step \mod policy\_update\_steps = 0$ **then**
            $\pi_{\boldsymbol{\theta}} \leftarrow \texttt{a2c}(\mathcal{D})$
            $\mathcal{D} \leftarrow \emptyset$
return: $\pi_{\boldsymbol{\theta}}$

solution space.

4.4.1 Reward Function

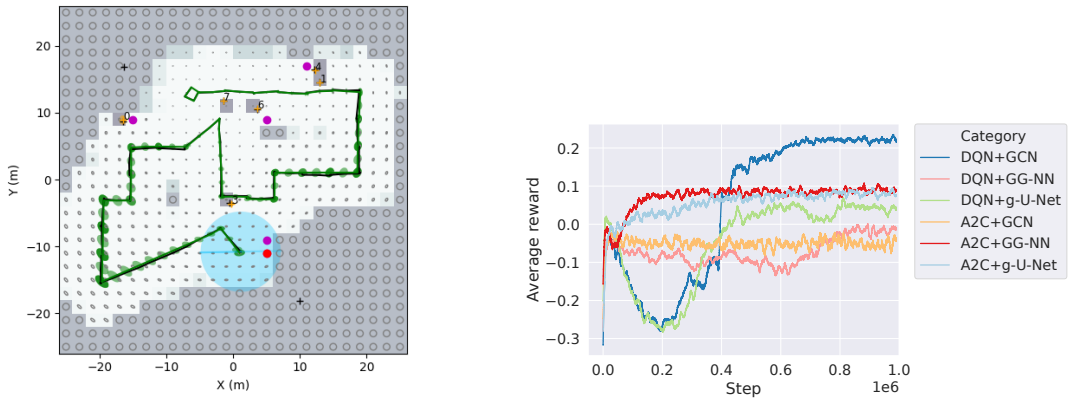
In Algorithm 4.1, we use linear normalization functions to map the range of the raw reward. If the optimal frontier is the one associated with the current pose, the maximum of the reward is 0, otherwise, it is 1 in all other cases. Algorithm 4.2 is designed based on the utility function of the EM exploration algorithm, Eq. (4.5), with an additional term penalizing the travel distance associated with an exploratory action. The raw reward is the difference in utility between the current state and

the subsequent state by taking the actions $\mathcal{U}$. The cost-to-go $C(\mathcal{U})$ with factor α encourages short travel distances in the raw reward function. The goal is to minimize this weighted combination of map uncertainty and travel distance with every given visit to a frontier node.

4.4.2 Exploration Training with RL

As shown in Algorithm 4.4, using an action sampling strategy, the robot chooses a next-view frontier based on an *exploration graph*. A reward and a new exploration graph are assigned after reaching the frontier most recently selected. The policy is trained using recorded experience data.

4.5 Experiments and Results



(a) An illustration of the simulation environment

(b) Average reward during the training process

Figure 4.2: The training performance of different methods on randomly generated simulation environments.

4.5.1 Experimental Setup

Our exploration policy is trained in a 2D simulation environment. The simulated robot has a horizontal field of view of 360° (±0.5°). The measurement range of the simulated robot is $5m$ ($\pm 0.02m$). While exploring, the robot can rotate from $-180°$ to 180° (±0.2°), and travel from $0m$ to $2m$ ($\pm 0.1m$) at each timestep. All simulated noise is Gaussian, and we give its standard deviation in the intervals above. Given the goal frontier location, the robot will first turn to the goal and then drive directly to the goal following a straight-line path. We use an occupancy grid map to describe the environment. Each occupancy map is $40m \times 40m$ with randomly sampled landmarks and random initial robot locations. The *feature density* of landmarks is 0.005 per m^2 in our experiment. To simplify the problem, landmarks are assumed passable, so obstacle avoidance is not needed.

The virtual map is made up of $2m \times 2m$ square map cells, with a $1m^2$ initial error covariance in each dimension for the virtual landmarks. We note that virtual landmarks are only used to evaluate the reward of Alg. 4.2, which trades map accuracy against travel expense. Meanwhile, the virtual landmarks are excluded from both SLAM factor graphs and the exploration graph. The exploration task will be terminated once 85% of a map has been observed.

The simulation environment is written in Python and C++, and our graph neural network models are trained using PyTorch Geometric [68]. Our code is freely available on Github[2] for use by others.

[2] `https://github.com/RobustFieldAutonomyLab/DRL_graph_exploration`

4.5.2 Policy Training

The training environments are generated with uniformly randomly sampled landmarks and initial robot locations. A training environment example is shown in Fig. 4.2(a); the uncertainty of virtual landmarks is represented by the error ellipses in each cell of the map. The blue circle represents the current field of view and the center point is the current robot location. The green error ellipses represent the uncertainty of past robot poses. The magenta points are frontier candidates, which comprise the current action space, and the red point is the selected next-view frontier. Landmarks are indicated by plus signs, which we plot in yellow once observed by the robot. Cells containing true landmarks are denoted as occupied in our map to mark their locations (ensuring a unique and non-trivial occupancy map is obtained from every exploration trial), but the landmarks are infinetesimal in size, and assumed not to present occlusions or collision hazards.

The exploration policy is trained by DQN and A2C with three different GNN models each. The performance of each approach is shown in Figure 4.2(b). For further use and study in exploration experiments, we select the policies that achieve the highest average reward for each reinforcement learning framework, which are DQN+GCN and A2C+GG-NN.

4.5.3 Computation Time

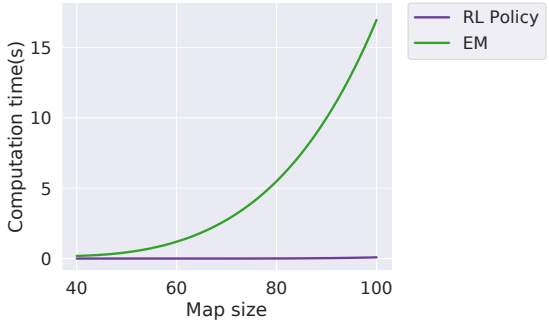
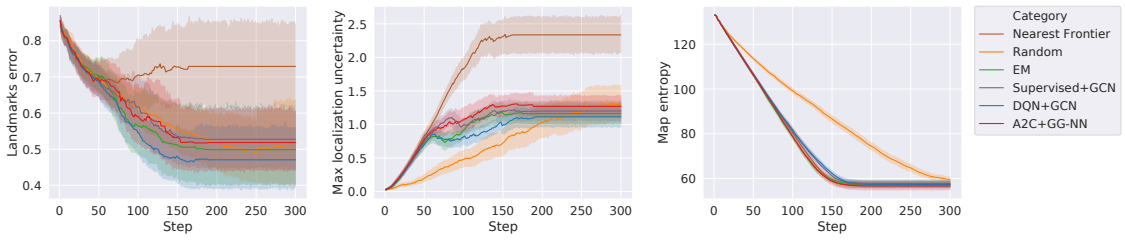


Figure 4.3: Computation time for exploration decision-making on different map sizes, which are square in the dimension indicated. Timing was evaluated on a computer equipped with an Intel i9 8-core 3.6Ghz CPU and an Nvidia GeForce Titan RTX GPU. The average computation time is 0.04427s for the RL policy.

The decision-making process is the most time-consuming component of exploration. The EM exploration algorithm uses forward simulation of a robot's SLAM process (including the propagation of uncertainty across virtual landmarks) to select the best next-view position from the available candidates. Hence the time complexity of the EM algorithm is $\mathcal{O}(N_{action}(C_1 + C_2))$, where N_{action} is the number of the candidate actions, C_1 is the cost of each iSAM2 predictive estimate over a candidate trajectory (a function of the number of mapped true landmarks and prior poses) and C_2 is the cost of the covariance update for virtual landmarks (a function of the number of prior poses and the number of virtual landmarks). We compare computation time between the EM algorithm and the RL policy on four different sizes of maps, namely $40m \times 40m$, $60m \times 60m$, $80m \times 80m$ and $100m \times 100m$. Each size has the same feature density, which is 0.005 landmarks per m^2. As shown in Fig. 4.3, the average computation time of the EM algorithm grows dramatically with increasing size of the map, which leads to larger action, state and belief spaces. The EM algorithm will ultimately prove unsuitable for real-time exploration in large, complex environments.

On the other hand, the RL policies use graph neural networks to predict the best action so that the computation time of the RL policy remains low, which meets the requirements for real-time application in high-dimensional state and action spaces.

4.5.4 Exploration Comparison

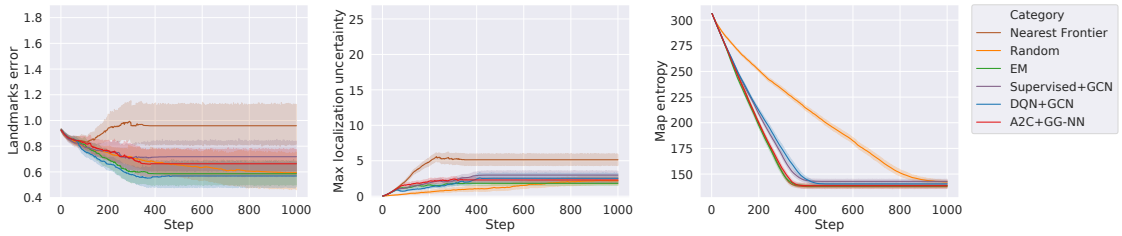


(a) Average uncertainty of landmarks

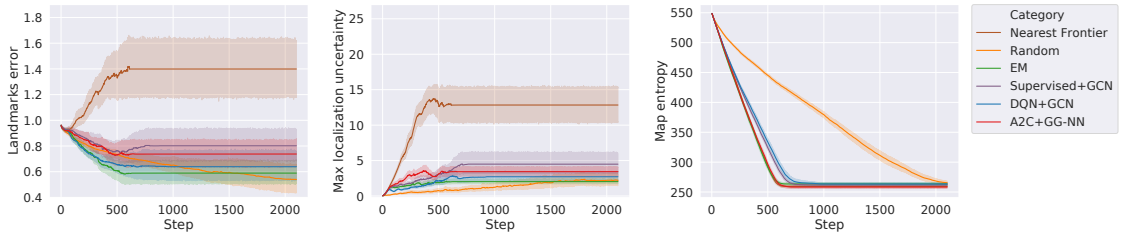
(b) Max uncertainty of the trajectory

(c) Map entropy reduction

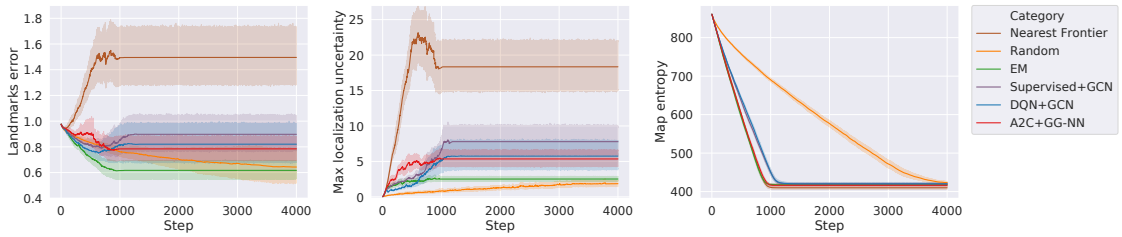
Figure 4.4: The result of 50 exploration trials of each method, with the same randomly initialized landmarks and robot start locations, on $40m \times 40m$ maps (the first three metrics shown are plotted per time-step of the simulation).



(a) Average uncertainty of landmarks (b) Max uncertainty of the trajectory (c) Map entropy reduction



(d) Average uncertainty of landmarks (e) Max uncertainty of the trajectory (f) Map entropy reduction



(g) Average uncertainty of landmarks (h) Max uncertainty of the trajectory (i) Map entropy reduction

Figure 4.5: The results of 50 exploration trials on large-size maps (each of which is larger than the $40m \times 40m$ maps used for training); (a), (b), (c) represent $60m \times 60m$ maps, (d), (e), (f) represent $80m \times 80m$ maps, and (g), (h), (i) represent $100m \times 100m$ maps.

We compared the learned policies with (1) a nearest frontier approach, (2) the selection of a random frontier, (3) the EM approach, and (4) the GCN model trained by supervised learning in [2] over 50 exploration trials. For each trial, every ap-

proach uses the same random seed to generate the same environment. Results of this comparison are shown in Figure 4.4, where we evaluate three metrics: the average landmark uncertainty (of real landmarks only), the maximum localization uncertainty along the robot's trajectory, and occupancy map entropy reduction. The EM algorithm offers the best performance, but is not real-time viable in high-dimensional state-action spaces. Both the Nearest Frontier method and the Random method are real-time viable methods. However, the Nearest Frontier method has the highest landmark and localization uncertainty, and the Random method has the lowest exploration efficiency. The learning-based methods can address real-time applications and offer low landmark and localization uncertainty. The Supervised+GCN method and DQN+GCN method have similar map entropy reduction rates, but DQN+GCN offers lower landmark and localization uncertainty because it travels longer distances from one iteration to the next by selecting high-value actions. In Fig. 4.4(c), both supervised+GCN and DQN+GCN have slower entropy reduction rates than EM and the A2C+GG-NN policy. When supervised+GCN does not perform with high accuracy, it will tend toward random action selection, because it is trained using binary labels.

Unlike supervised learning, the RL policies are seeking the largest value for each step. Thus, when DQN+GCN does not have high accuracy, it will choose the highest-value action based on predicted values. Although the DQN+GCN policy has relatively low accuracy in predicting the highest-value action, it still selects high-value actions instead of random ones, which leads to more accurate mapping. The A2C+GG-NN policy has the highest exploration efficiency among learning approaches, and offers lower landmark uncertainty than the Supervised+GCN method, but not lower than the DQN+GCN method. Representative exploration trials for all of the algorithms compared in Figure 4.4 are provided in our video attachment.

4.5.5 Scalability

During testing, it is possible to encounter large-scale environments that induce a lengthier pose history, more landmarks, and more frontiers, generating larger exploration graphs over longer-duration operations than a robot may have been exposed to during training. We demonstrate here that our RL policies can be trained in small environments, and scale to large environments to perform exploration tasks. In this experiment, our RL policies are trained on $40m \times 40m$ maps with 0.005 landmarks per m^2 feature density. The size of the testing environments are $60m \times 60m$, $80m \times 80m$ and $100m \times 100m$ with the same feature density.

The exploration results are shown in Fig. 4.5. The A2C+GG-NN method has nearly the same exploration efficiency as the EM algorithm and the Nearest Frontier approach, which all explore faster than the Supervised+GCN and DQN+GCN methods as shown in Figs. 4.5(c), 4.5(f), and 4.5(i). The landmark uncertainty (Figs. 4.5(a), 4.5(d), 4.5(g)) and localization uncertainty (Fig. 4.5(b), 4.5(e), 4.5(h)) of our learning-based methods gradually degrade in performance with increasing map size. The Supervised+GCN method accumulates the largest landmark and localization uncertainty among learning-based methods for each map size. DQN+GCN achieves better results over $60m \times 60m$ and $80m \times 80m$ maps than A2C+GG-NN, but the performance of A2C+GG-NN is better than DQN+GCN over $100m \times 100m$ maps because of the overfitting of DQN+GCN. Overall, A2C+GG-NN demonstrates the best scalability performance based on its high exploration efficiency and relatively low landmark and localization uncertainty across all trials. Representative exploration trials showing the performance of A2C+GG-NN across all of the map sizes examined are provided in our video attachment.

4.6 Conclusions

We have presented a novel approach by which a mobile robot can learn, without human intervention, an effective policy for exploring an unknown environment under localization uncertainty, via exploration graphs in conjunction with GNNs and reinforcement learning. The exploration graph is a generalized topological data structure that represents the states and the actions relevant to exploration under localization uncertainty, and we build upon our previous work that used them for supervised learning with GNNs.

Through our novel integration of this paradigm with reinforcement learning, a robot's exploration policy is shaped by the rewards it receives over time, and a designer does not need to tune hyperparameters manually, as is the case for supervised learning. Additionally, the policy learned from RL algorithms is non-myopic and robust across a variety of test environments. Our learned RL policies have exhibited the best performance among the real-time viable exploration methods examined in this chapter. Moreover, we have shown that policies learned in small-scale, low-dimensional state-action spaces can be scalable to testing in larger, higher-dimensional spaces. The RL policies learned offered high exploration efficiency and relatively low map and localization uncertainty among the real-time viable exploration methods examined, across the various examples explored.

Chapter 5

Zero-Shot Reinforcement Learning for Autonomous Exploration

In this chapter, we aim to produce zero-shot reinforcement learning for more realistic exploring mobile robots gathering 3D lidar observations, using the same SLAM framework as in [12] to support exploration. Coupling the GNN-based DRL model with this SLAM framework, the learned policy successfully achieves zero-shot transfer in both virtual and real environment tests.

5.1 Problem Formulation

5.1.1 Simultaneous Localization and Mapping Framework

An illustration of the SLAM factor graph is shown in Fig. 5.1(a). There are two types of sequential factors; blue factors indicate the *odometry* measurements (ϕ^O) between two consecutive poses, and green factors represent *sequential scan matching* constraints (ϕ^{SSM}). These are provided by the iterative closest point (ICP) algorithm, using 3D LiDAR point clouds to estimate the relative transformation between consecutive poses. There are also two types of loop closure constraints. *Pose matching* (ϕ^{PM}), indicated by magenta in Fig. 5.1(a), provides a loop closure constraint when the point cloud from the current pose has been successfully matched with the point cloud from a previous pose. Finally, *segment matching* (ϕ^{SM}), colored by red in Fig. 5.1(a), achieves a loop closure when two poses observe the same segmented object in their respective point clouds (using the approach of [69]).

The motion model and the measurement model of our SLAM framework are

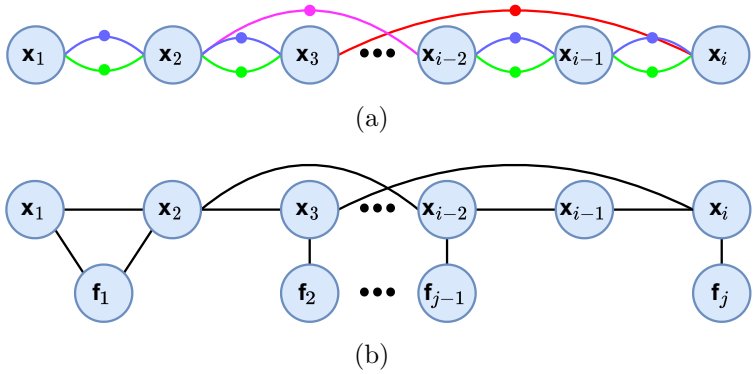


Figure 5.1: **An illustration of the SLAM factor graph and the exploration graph.** Top: The SLAM factor graph contains four different types of constraints: blue factors are provided by odometry measurements; green factors are obtained from sequential scan matching of two consecutive poses; red factors represent the loop closures provided by point cloud segment matching; magenta factors are loop closures generated by pose matching. Bottom: The corresponding exploration graph is shown, containing all poses from the SLAM factor graph, and waypoints representing map frontiers. If two poses have a constraint joining them in the SLAM factor graph, an edge will be assigned to connect these two poses. The current pose x_i is connected to the nearest frontier, and any frontiers whose paths achieve place revisiting are connected to the prior poses they revisit. All the edges in the exploration graph are weighted with Euclidean distances.

defined as:

$$\mathbf{x}_i = h_i(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}_i, \quad \mathbf{w}_i \sim \mathcal{N}(\mathbf{0}, Q_i), \tag{5.1}$$

$$\mathbf{z}_k = g_k(\mathbf{x}_{i_k}) + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, R_k), \tag{5.2}$$

where $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^t$ are 6-DOF robot poses and $\mathcal{U} = \{\mathbf{u_i}\}_{i=1}^t$ is a given motion input.

Then we define the factor graph:

$$\phi(\mathbf{x}) = \phi^0(\mathbf{x}_0) \prod_i \phi_i^{\mathrm{O}}(\mathbf{x}_i) \prod_j \phi_j^{\mathrm{SSM}}(\mathbf{x}_j) \qquad \text{(sequential)}$$

$$\prod_p \phi_p^{\mathrm{PM}}(\mathbf{x}_p) \prod_q \phi_q^{\mathrm{SM}}(\mathbf{x}_q). \qquad \text{(loop closures)}$$

The SLAM problem then becomes a nonlinear least-squares optimization problem on a factor graph. We use iSAM2 [64] to solve this problem.

We adopt the *virtual map* framework from the EM exploration algorithm of [11]. A virtual map is uniformly discretized at the same or lower resolution than the robot's occupancy map, and contains *virtual landmarks*, $\tilde{\mathbf{l}}_k \in \tilde{\mathcal{L}}$, populating the map's cells. Each virtual landmark has a large initial covariance, which will be driven down by the robot as it observes the contents of the map cell. In this setting, the goal of exploration is to minimize the covariance of all *virtual landmarks*, which leads a robot to both explore efficiently and to produce an accurate map. The utility function of the current state is defined as follows:

$$U(\tilde{\mathcal{L}}) = \sum_{\tilde{\mathbf{l}}_k \in \tilde{\mathcal{L}}} \log\det(\Sigma_{\tilde{\mathbf{l}}_k}), \tag{5.3}$$

where $\Sigma_{\tilde{\mathbf{l}}_k}$ is the covariance matrix for virtual landmark $\tilde{\mathbf{l}}_k$.

5.1.2 Exploration Graph for Real Environments

We define the exploration graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as shown in Fig. 5.1(b). There are two types of vertices in $\mathcal{V}$. $\mathcal{X} \subset \mathcal{V}$ contains the robot pose history. Poses connected by constraints in the SLAM factor graph in Fig. 5.1(a) are also connected with edges in the exploration graph. Furthermore, the exploration graph includes exploration waypoints derived from map frontiers, $\mathcal{F} \subset \mathcal{V}$. These frontier nodes are extracted from the map's boundaries between free and unknown areas. The current pose x_t is connected with the *nearest* frontier to its location. If a frontier can provide place-revisiting through either pose matching or segment matching, we connect the previous poses associated with that loop closure to this frontier. All other frontiers, which are neither the nearest frontier to the current pose, nor achieve place-revisiting, are

excluded from the exploration graph. All edges in the graph are weighted by their Euclidean distances.

Each vertex $\mathbf{n_i} \in \mathcal{V}$ has a feature vector:

$$\mathbf{s}_i = [s_{i_1}, s_{i_2}, s_{i_3}, s_{i_4}],$$

$$s_{i_1} = \phi_{\mathrm{A}}(\Sigma_i), \tag{5.4}$$

$$s_{i_2} = \sqrt{(x_i - x_t)^2 + (y_i - y_t)^2}, \tag{5.5}$$

$$s_{i_3} = \arctan2(y_i - y_t, x_i - x_t), \tag{5.6}$$

$$s_{i_4} = \begin{cases} 0 & \mathbf{n_i} = \mathbf{x_t} \\ 1 & \mathbf{n_i} \in \{\mathbf{f_n}\} \\ -1 & \text{otherwise} \end{cases}. \tag{5.7}$$

In Eq. (5.4), we adopt the A-Optimality criterion as a metric to evaluate the uncertainty level of each node. The covariances of frontier vertexes $\mathcal{F}$ are extracted from the virtual map. We capture the geometric information of the current robot state using Eqs. (5.5) and (5.6), which contain the relative distance and orientation information between current pose x_t and the node n_i. The last feature s_{i_4} is used to indicate the identity of each node. The current pose is labeled as 0, all previous poses are -1, and frontiers are 1.

5.2 Algorithms and System Architecture

5.2.1 Graph Neural Networks

In this chapter, we use two Graph U-Nets (g-U-Nets) [70] to serve as the policy network and the value network, respectively. Similar to U-Net [71], g-U-Nets have graph pooling layers to encode the input feature vectors of nodes in the input graph.

Additionally, the graph unpooling layers are used to decode the graphs in the hidden layers to provide the output graphs. Besides the pooling and unpooling layers, each encoder and decoder has a Graph Convolutional Network (GCN) [20] layer to update the graph features. The depth of our g-U-Nets is 3 and the number of features for each hidden layer is 1000. A multilayer perceptron (MLP) output layer is adopted to provide the final output prediction. A dropout layer with a 0.5 dropout rate is placed between the output of our g-U-Nets and the MLP output layer.

5.2.2 Deep Reinforcement Learning

In our framework, the robot is solving a decision-making problem over exploration graphs using a learned policy from DRL. The candidate actions are paths from the current pose to the frontier nodes in the exploration graph. The exploration graph contains information about all past poses, their respective uncertainty, and how they relate to map frontiers. At each decision-making instant k, the state of our system is represented by an exploration graph $\mathcal{G}_k \in \mathsf{G}$. A reward $R_k \in \mathbb{R}$ is assigned to the selected action $\mathbf{f}_k \in \mathsf{F}_{\mathcal{G}_k}$. The overall decision-making process can be modeled as a Markov Decision Process $\langle \mathsf{G}, \mathsf{F}, \Pr, \gamma \rangle$ [66].

We consider the A2C [67] policy-based DRL algorithm in this chapter, for which two separate g-U-Nets serve as the policy network and the value network. The loss function is defined as follows:

$$
\begin{aligned}
L_{\mathrm{A2C}}(\mathcal{D}) &= \mathbf{E}_{\mathcal{B}\sim\mathcal{D}}[L^{(1)}_{\mathrm{A2C}} + \eta L^{(2)}_{\mathrm{A2C}}], \\
L^{(1)}_{\mathrm{A2C}} &= [A(\mathcal{G}, \mathbf{f}) - V(\mathcal{G}))\log \pi(\mathbf{f}|\mathcal{G}) + \beta(A(\mathcal{G}, \mathbf{f})]^2, \\
L^{(2)}_{\mathrm{A2C}} &= \sum_{\mathbf{f}\in\mathsf{F}_{\mathcal{G}}} \pi(\mathbf{f}|\mathcal{G}) \log \pi(\mathbf{f}|\mathcal{G}),
\end{aligned}
\tag{5.8}
$$

where the *advantage* function is defined as $A(\mathcal{G}, \mathbf{f}) = Q(\mathcal{G}, \mathbf{f}) - V(\mathcal{G})$ to evaluate the

Algorithm 5.1: Reward Function

input: Exploration graph $\mathcal{G}$, Frontier node $\mathbf{f}$
\# Calculate the raw reward (Eq. 5.9)
$\mathcal{R}^0_{\mathcal{G}} = U^0(\tilde{\mathcal{L}}) - U'_{\mathcal{U}}(\tilde{\mathcal{L}}') - \alpha C(\mathcal{U})$
\# Normalize the raw reward
$\mathcal{R}_{\mathcal{G}} = \{r_{\mathbf{f}'} = \mathcal{R}^0_{\mathcal{G}}\}$
$l = \min \mathcal{R}_{\mathcal{G}}$, $u = \max \mathcal{R}_{\mathcal{G}}$
$r_{\mathbf{f}} \leftarrow (r_{\mathbf{f}} - l)/(u - l)$
\# Compute projection based on nearest frontier
$\mathbf{f}_t = \texttt{nearest\_frontier}(\mathbf{x}_t)$
if $u =$ *raw_reward*$(\mathbf{f}_t)$ **then**
  return $r_{\mathbf{f}} - 1$ \# $r(\mathcal{G}, \mathbf{f}) \in [-1, 0]$
return $2r_{\mathbf{f}} - 1$ \# $r(\mathcal{G}, \mathbf{f}) \in [-1, 1]$

difference between the state value and the state-action value. $\beta \in \mathbb{R}$ is a coefficient for the loss of the value function. The entropy coefficient $\eta \in \mathbb{R}^+$ is used to weigh output entropy for encouraging exploration during training.

5.2.3 Reward Function

We use the utility function given in Eq. (5.3) from the EM exploration algorithm [11], whose behavior we wish to emulate, to compute the raw reward for actions that travel to each candidate frontier. The raw reward is defined as follows:

$$\mathcal{R}^0_{\mathcal{G}} = U^0(\tilde{\mathcal{L}}) - U'_{\mathcal{U}}(\tilde{\mathcal{L}}') - \alpha C(\mathcal{U}), \tag{5.9}$$

where $\mathcal{U}$ contains sequential actions to the selected frontier position. The output of cost-to-go function $C(\mathcal{U})$ is the travel distance weighed by coefficient α, expressing a preference for shorter paths. We then set a new range for these raw rewards by linear normalization. If the frontier selected by the EM algorithm is the *nearest* frontier associated with the current pose, the range of the reward is $[-1, 0]$. Otherwise, the

range is $[-1, 1]$. The reward function is described in Alg. 5.1.

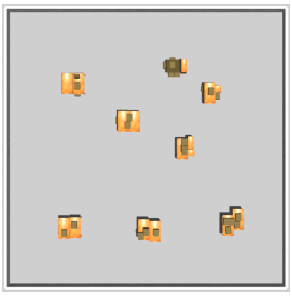


Figure 5.2: The office-like Gazebo environment used for training.

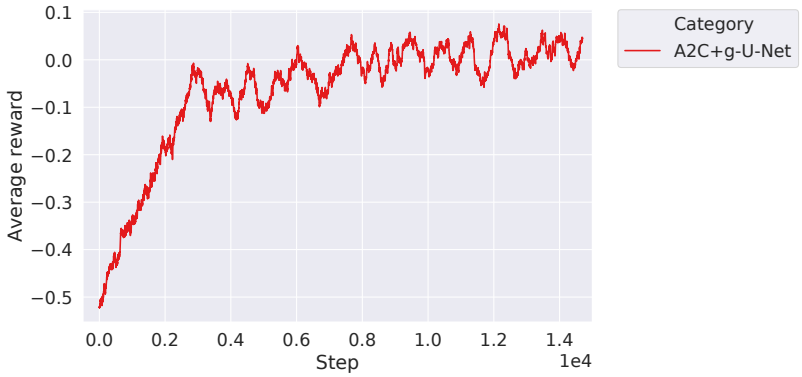


Figure 5.3: The average reward during training.

5.2.4 Computational Complexity

For the EM algorithm, the computational complexity of the decision-making process is $\mathcal{O}(N_{\text{actions}}(C_1 + C_2))$, where $C_1 = \mathcal{O}(n^3)$ bounds the complexity of the iSAM2 update (n is the number of poses), and $C_2 = \mathcal{O}(m)$ bounds the covariance update for virtual landmarks (m is the number of virtual landmarks) [11]. The computation time increases significantly in the size of the state-action space, limiting the framework's applicability. On the other hand, as shown in [3], the computation time for decision-making with our fully trained DRL GNN framework is nearly constant, allowing real-time performance across a wide range of problems.

5.3 Experiments and Results

5.3.1 Experimental Setup

We assume that an unmanned ground vehicle (UGV) must explore an indoor environment populated with 3D obstacles, it relies upon 6 degree-of-freedom SLAM, and it is permitted to reason about exploration with respect to the ground plane, where belief space planning is performed. The robot uses a 2D map to perform three degree-of-freedom motion planning reach the goal position. The simulation environments used in this work are built in Gazebo and explored using the Robot Operating System (ROS). A simulated Clearpath Jackal robot is equipped with wheel odometry and a VLP-16 3D LiDAR. We restrict the sensor range of our LiDAR to 3 meters to induce challenging pose uncertainty in our exploration comparison. We adopt the default noise settings in Gazebo for the simulated sensors, and we manually add Gaussian noise to robot translation and rotation actions of standard deviation 0.01m and 0.08°, respectively. We use the same SLAM framework employed in [12], with the GTSAM library [63]. Dijkstra's algorithm is used to generate a path to each frontier waypoint. A 2D map of virtual landmarks is maintained in the ground plane to provide the reward for each decision-making selection during the DRL policy training. The resolution of the map is 0.5m per cell and the standard deviation of the initial covariance of each virtual landmark is 0.2m. We terminate the exploration task once 85% of the environment's ground plane has been explored. Additionally, we keep track of the robot's volumetric sensor coverage of the environment using a separate 3D occupancy map of the workspace.

Our graph neural network models are trained using PyTorch Geometric [68]. The desktop used for policy training and simulation testing is equipped with an Intel i9 3.6Ghz CPU and an Nvidia Geforce Titan RTX GPU. For real-world exploration

experiments (testing only), our algorithms run on a Dell Precision 3541 mobile workstation laptop which has an Intel Xeon CPU and an Nvidia Quadro P620 GPU.

5.3.2 Policy Training

To evaluate the transferability of our proposed approach, we only use one environment during training. The office-like training environment is shown in Fig. 5.2. We randomly placed 8 objects in this environment. The robot has nine fixed initial locations in this environment from which it begins exploring.

We apply 15,000 training episodes in total, which, motivated by the expense of our ROS/Gazebo simulation, is orders of magnitude fewer than in our prior work with 2D landmark-based SLAM [3]. We set the learning rate to 0.0001 and perform a policy update every 10 steps. The average reward obtained during training is shown in Fig. 5.3. Throughout this process, the state of the system is represented by the exploration graph, and the action space is comprised of the frontier nodes in the exploration graph.

Figure 5.4: The Clearpath Jackal UGV used in our experiments.

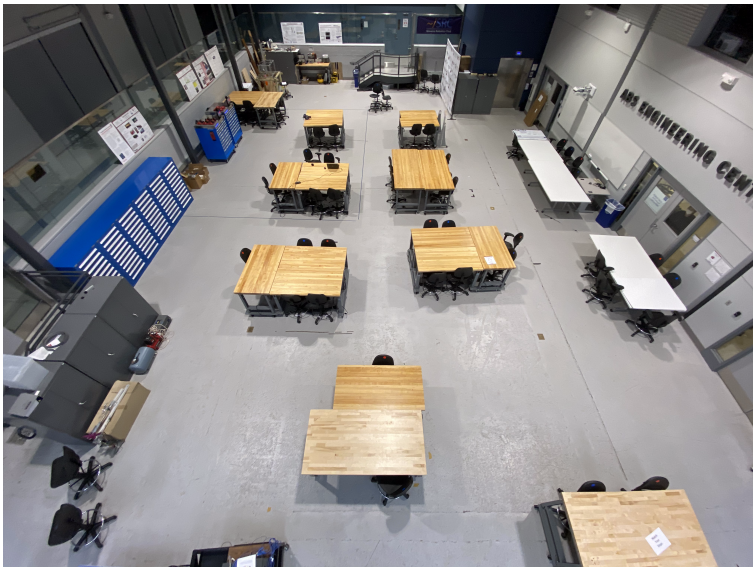
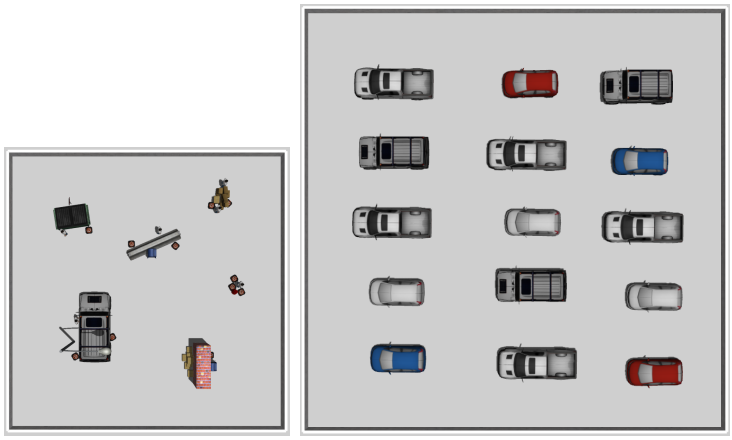


Figure 5.5: Environment used for real-world deployment of our UGV.

5.3.3 Simulated Exploration Comparison

We use the learned policy trained in the office-like environment to test in two different environments. In Fig. 5.6(a), we build a "street environment" which has the same $20m \times 20m$ size as the training environment. However, this street environment contains fewer objects. Also, the size and the shape of these objects differ from the training environment. The exploration results are shown in Fig. 5.7(a) and 5.7(b). We compare the learned graph-based policy with (1) a nearest frontier (NF) approach [30], (2) the EM algorithm [11, 12], and (3) a random frontier selection approach over 10 exploration trials. For each trial, we always initialize the robot from the center of the testing environment. We compute the mean absolute error between the mapped 3D points associated with the estimated robot trajectories and the ground truth trajectories to determine the map error. We also compute the total 3D volume mapped during each trial.



(a) (b)

Figure 5.6: Gazebo simulation testing environments.

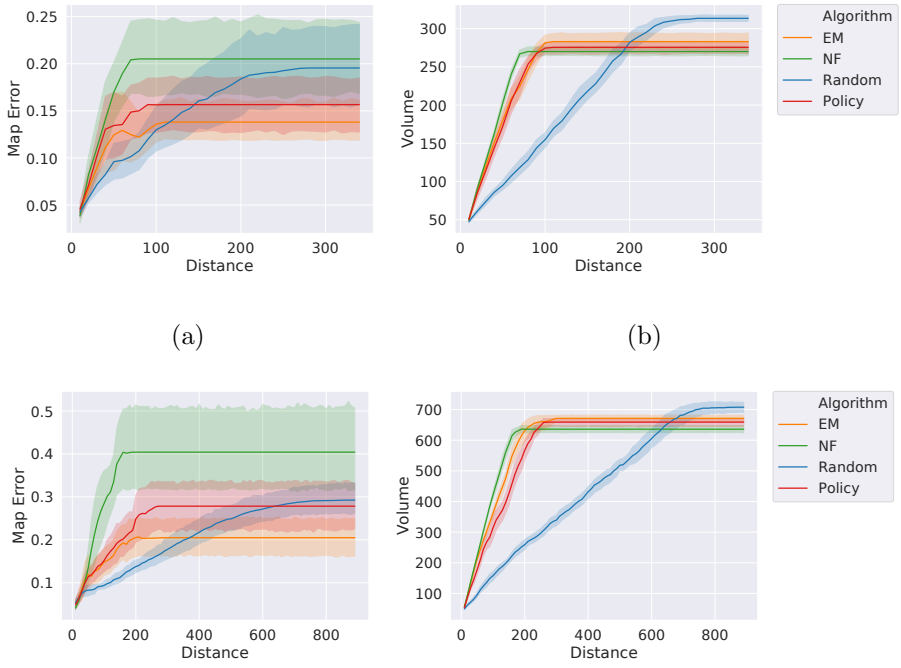
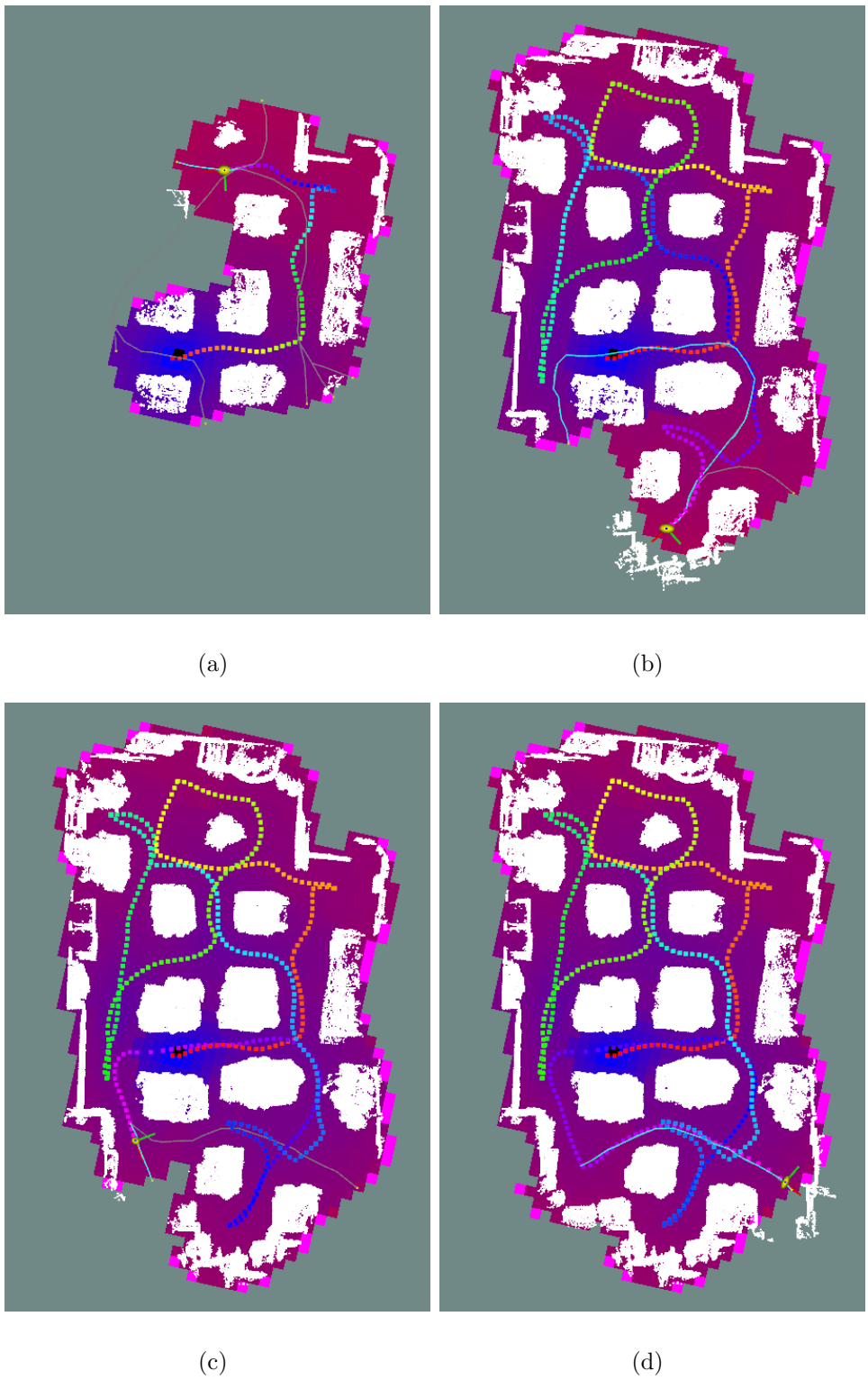


Figure 5.7: Autonomous exploration results from our Gazebo simulation testing environments.

The EM algorithm achieves the lowest map error in the end, but has a slightly

worse exploration efficiency than the NF approach. Although the NF approach is the most efficient method for covering an unknown environment, it offers the worst map accuracy during exploration because it achieves the fewest loop closures. The random method achieves the most loop closures during exploration, but its long travel distances generate a large accumulated error that cannot be completely eliminated by these loop closures. It also covers the environment very inefficiently. Our learned policy achieves a very similar exploration efficiency to the EM algorithm, and its map accuracy is surpassed only by EM algorithm, whose performance we seek to emulate.

The second simulation testing environment is shown in Fig. 5.6(b). In this "parking garage" environment, there are fifteen evenly spaced cars and the size of this environment is $30m \times 30m$, which is larger than the training environment. Like the first simulation environment, each exploration algorithm has 10 trials initialized from the center of the environment. The learned policy has a slightly worse exploration efficiency than the EM algorithm in this large environment, but it once again achieves the second lowest map error (again, second to the EM algorithm) compared with other exploration methods. All other algorithms have the same relative performance as in the "street" environment.



(a) (b)

(c) (d)

Figure 5.8: Ground-plane map and trajectory resulting from our graph-based policy's exploration of the Stevens ABS Engineering Center.

5.3.4 Real-world Exploration

Our learned policy can also be transferred to a real-world environment. Our Clearpath Jackal UGV, shown in Fig. 5.4, is equipped with the same odometry and LiDAR sensing as the simulated UGV. We test our learned policy in the ABS Engineering Center at Stevens Institute of Technology shown in Fig. 5.5. In Fig. 5.8, we present an example of autonomous exploration using the learned, graph-based policy. The robot pose is represented by red-green axes, and the yellow ellipsoid indicates the uncertainty of the current pose. The cyan path is the path to the selected frontier, and gray paths are for other unselected frontiers. The cost map covering the ground plane represents the uncertainty of the current virtual map, where blue color indicates the lowest uncertainty. Magenta represents the high uncertainty of our virtual map prior. We terminate the exploration task if there are no frontiers detected on the current map. Although our exploration algorithm has real-time computation, there is a short-term pause in each decision-making step because of the computational expense of SLAM and the required update of occupancy maps and virtual maps. In Fig. 5.8(a), the robot chooses the nearest frontier to explore the environment, rather than the longer-distance path to obtain a loop closure, because this is the beginning of the exploration and a loop closure only reduces the uncertainty of a small portion of the current virtual map. In Fig. 5.8(b), the robot selects a longer-distance path to obtain a loop closure to reduce the uncertainty of the current virtual map. In Fig. 5.8(c), the uncertainty of the right bottom area on the map is reduced by taking the selected path. We present the final exploration result in Fig. 5.8(d). The overall exploration process is shown in our video attachment.

5.4 Learning Exploration from A Portfolio of Algorithms

In the previous sections, the learned exploration policy was trained by the EM algorithm. However, the EM algorithm is not always the best exploration method in any given environment. For example, the nearest frontier (NF) approach achieves the best exploration performance in the environment shown in Fig. 5.9, which is empty in its center. As shown in Fig. 5.10(a) and Fig. 5.10(b), the NF method offers better exploration performance than the EM algorithm because localization uncertainty and map error cannot be reduced by the place revisiting process in this particular environment, due to the lengthy travel required for place recognition. Therefore, The nearest frontier (NF) method provides the best exploration performance in the new training environment.

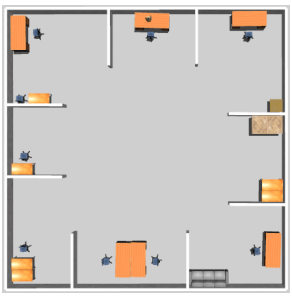
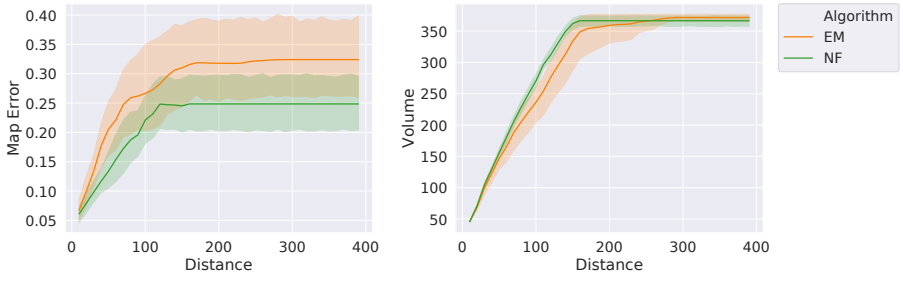


Figure 5.9: The empty-center Gazebo environment used for learning the NF strategy.



(a) (b)

Figure 5.10: Autonomous exploration results from the new empty-center Gazebo simulation training environment.

5.4.1 Reward Function

The policy should imitate the NF exploration strategy in the given environment. The raw reward is generated by the NF utility, which is defined as follows:

$$\mathcal{R}^0_{\mathcal{G}} = C(\mathcal{U}), \tag{5.10}$$

where $\mathcal{U}$ contains sequential actions to the selected frontier position and $C(\mathcal{U})$ is the travel cost function. We use the same normalization method as in 5.2.3 to generate the reward for training. The maximum reward is from the shortest travel distance.

5.4.2 Policy Training

The simulation setting and the robot are the same as those described in Sec. 5.3. We use a new empty-center Gazebo simulation environment shown in Fig. 5.9 to add the NF strategy to the learned policy in Sec 5.3.2. We use the learned policy trained by the EM algorithm as the initial policy. Because the feature NF method is easy to learn, overfitting problems will occur if we train our new policy with many training steps. To train a policy to learn the NF method, 500 additional training episodes are used. The robot starts from the center position of the environment during the training process. We still use the A2C RL algorithm with a 0.0001 learning rate and a policy update rate of 10 to learn the new policy.

5.4.3 Exploration Performance

For this study, we have built a new testing environment that is $30m \times 40m$ in size. The environment, illustrated in Fig. 5.11, has a similar structure to the training environment of Fig. 5.9, but the objects have different shapes and quantities. Fig. 5.12(a) and Fig. 5.12(b) show the exploration results in the new testing environment.

We compare the new learned policy with (1) an NF approach, (2) the EM algorithm, and (3) the previous policy trained by the EM method. Each method has 10 exploration trials. In Fig. 5.12(a), the EM algorithm has the largest map error. The policy trained by the EM algorithm is better than the EM algorithm because the learned policy loses the estimation accuracy in this large map, which happens to produce a better map result than the original EM algorithm. The new policy has nearly the same low map error as the NF approach. In Fig. 5.12(b), the policy trained by the EM algorithm has the lowest exploration efficiency because it loses its prediction accuracy. The NF approach has the highest exploration efficiency because it always chooses the shortest path to explore. The performance of the EM algorithm and the new policy is similar, and they lie between the NF approach and the previous policy.

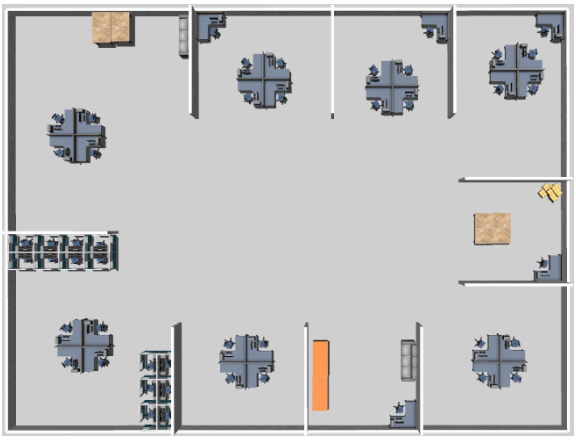


Figure 5.11: The Gazebo environment used for testing.

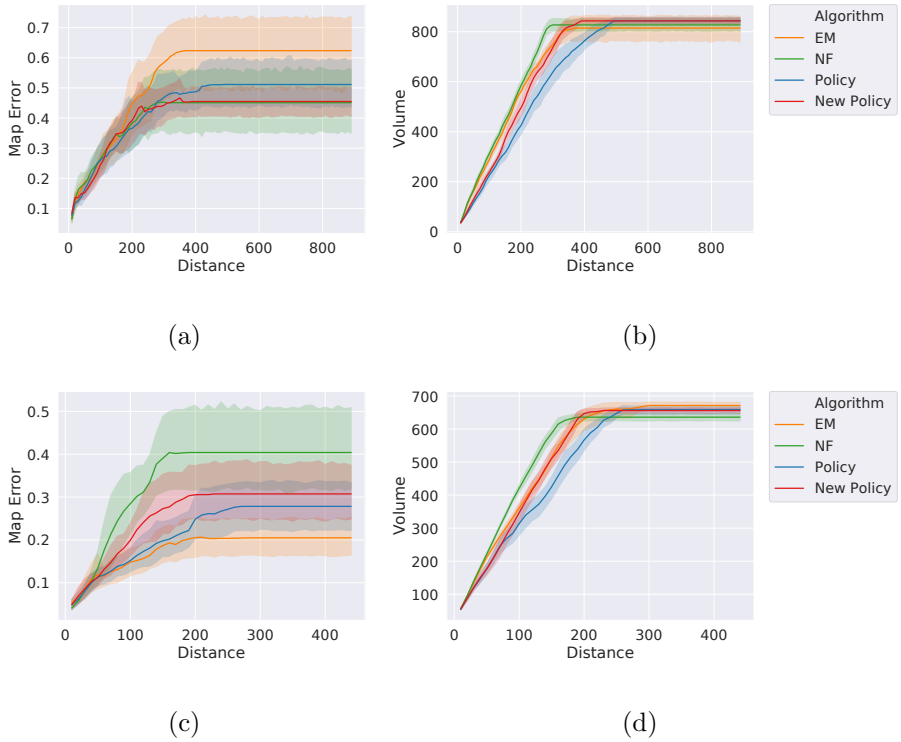


Figure 5.12: Autonomous exploration results from Gazebo simulation testing environments. The results at top correspond to the environment shown in Fig. 5.11, and the results at bottom correspond to the environment shown in Fig. 5.6(b).

To prove that our new policy can still use the strategy learned from the EM algorithm, we run 10 exploration test trials in the previous testing environment shown in Fig. 5.6(b). In Fig. 5.12(c), the new policy has slightly worse performance than the previous policy, but the difference is acceptable. In Fig. 5.12(d), the exploration efficiency is close to the EM algorithm and better than the previous policy.

5.5 Conclusions

In this chapter, we present a zero-shot transfer learning framework for mobile robot exploration under uncertainty that leverages an exploration graph as an efficient abstraction of a robot's state and environment. We have enhanced the DRL GNN

framework developed in our prior work [3] so it can be applied, for the first time, to the exploration of environments populated with complex obstacles, perceived using dense 3D range observations. Successful training now depends on high-fidelity simulation, and accordingly, our approach offers a highly efficient training process to meet these requirements; the exploration policy is trained in a single virtual environment and is successfully transferred to both virtual and real environments containing different obstacle quantities, arrangements, and geometries. The exploration graph proposed in this work offers generality that is suitable for a wide variety of real-world exploration tasks. The uncertainty of the traversability information can be embedded within the edges of the exploration graph to predict the uncertainties and hazards of the current environment.

Moreover, a new exploration policy learns from a portfolio of algorithms to explore different environments. We extend the previous policy trained by the EM algorithm to learn a new NF exploration strategy for a specific environment structure without forgetting the previous EM strategy training. Hence, the new learned policy is capable of choosing an optimal exploration strategy according to the current state of the environment.

Chapter 6
Conclusions and Future Work

6.1 Conclusions

The objective of this thesis is to achieve high-performance real-time decision-making during complex instances of mobile robot exploration, using deep learning models. The exploration tasks are difficult to learn because the test environments are unknown, so the robot needs to learn the exploration strategies through training in environments with relevant characteristics. The proposed exploration graph is an effective state representation for a DRL model. With GNNs, a mobile robot can learn high-performance exploration strategies through a set of training environments which differ greatly from the testing environments. The GNN-based RL policy offers appealing generalizability, which means the learned policy can be used in a wide range of state-action spaces, as long as the exploration graph topology, and key characteristics of the robot and its SLAM process, are similar to those of the training environments.

Moreover, one of the proposed learning models offers zero-shot transfer performance – policies learned from a simulation environment are successfully transferred to both virtual and real environments that differ from the training environment. Additionally, the proposed system can learn from a portfolio of exploration algorithms, so the learned policy is able to choose the best exploration strategy for a specific test environment.

6.2 Future Work

There are two keys areas that comprise promising topics for future inquiry related to the focus of this thesis.

6.2.1 Reinforcement Learning for Learning from a Portfolio of Algorithms

Although learning a new RL policy atop an existing policy can combine two exploration methods into one unified policy, achieving the best possible efficiency and performance when training a policy that combines multiple strategies remains an open problem. Future work may benefit from considering the use of curricula in reinforcement learning methods; [72], [73], [74] are able to help train a policy for multi-task purposes. Moreover, the Asynchronous Advantage Actor Critic (A3C) [28] reinforcement learning model can train a multi-task policy similar to ours via parallel training with multiple actors.

6.2.2 SLAM Graph Optimization by Learning Methods

The decision-making process for autonomous mobile robot exploration is nearly real-time viable when aided by neural networks as proposed in this work. However, the optimization process required by most SLAM frameworks is still time-consuming, and is not real-time viable over dense observations and lengthy time-scales. A collaboration with Yewei Huang, a fellow lab member of RFAL, has been looking into learning the correct localization results from SLAM training data produced in simulation environments. Future work will train graph neural network (GNN) models to estimate the result for the SLAM graph optimization process, so the overall exploration process can meet needs of real-time operation in any given environment.

Bibliography

[1] 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*, p. 0396, 2019.

[2] F. Chen, J. Wang, T. Shan, and B. Englot, "Autonomous exploration under uncertainty via graph convolutional networks," in *International Symposium on Robotics Research*, 2019.

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

[4] F. Chen, P. Szenher, Y. Huang, J. Wang, T. Shan, S. Bai, and B. Englot, "Zero-shot reinforcement learning on graphs for autonomous exploration under uncertainty," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[5] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[6] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[7] F. Dellaert, M. Kaess, *et al.*, "Factor graphs for robot perception," *Foundations and Trends in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017.

[8] R. Valencia, J. V. Miró, G. Dissanayake, and J. Andrade-Cetto, "Active pose slam," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1885–1891, 2012.

[9] H. Carrillo, I. Reid, and J. A. Castellanos, "On the comparison of uncertainty criteria for active slam," in *IEEE International Conference on Robotics and Automation*, pp. 2080–2087, 2012.

[10] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using rao-blackwellized particle filters.," in *Robotics: Science and Systems*, vol. 2, pp. 65–72, 2005.

[11] J. Wang and B. Englot, "Autonomous exploration with expectation-maximization," in *International Symposium on Robotics Research*, 2017.

[12] J. Wang, T. Shan, and B. Englot, "Virtual maps for autonomous exploration with pose slam," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4899–4906, 2019.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, pp. 396–404, 1990.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

[17] S. El Hihi and Y. Bengio, "Hierarchical recurrent neural networks for long-term dependencies," in *Advances in Neural Information Processing Systems*, pp. 493–499, 1996.

[18] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[19] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[21] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *International Conference on Learning Representations (ICLR)*, 2016.

[22] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *International Conference on Learning Representations (ICLR)*, 2018.

[23] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[24] 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.

[25] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, pp. 1995–2003, 2016.

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

[27] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[28] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, pp. 1928–1937, 2016.

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

[30] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pp. 146–151, 1997.

[31] B. Charrow, G. Kahn, S. Patil, S. Liu, K. Goldberg, P. Abbeel, N. Michael, and V. Kumar, "Information-theoretic planning with trajectory optimization for dense 3d mapping.," in *Robotics: Science and Systems*, vol. 11, 2015.

[32] B. Charrow, S. Liu, V. Kumar, and N. Michael, "Information-theoretic mapping using cauchy-schwarz quadratic mutual information," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4791–4798, 2015.

[33] B. J. Julian, S. Karaman, and D. Rus, "On mutual information-based control of range sensing robots for mapping applications," *The International Journal of Robotics Research*, vol. 33, no. 10, pp. 1375–1392, 2014.

[34] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[35] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[36] M. G. Jadidi, J. V. Miró, R. Valencia, and J. Andrade-Cetto, "Exploration on continuous gaussian process frontier maps," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6077–6082, 2014.

[37] M. G. Jadidi, J. V. Miro, and G. Dissanayake, "Mutual information-based exploration on continuous occupancy maps," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6086–6092, 2015.

[38] S. Bai, J. Wang, K. Doherty, and B. Englot, "Inference-enabled information-theoretic exploration of continuous action spaces," in *International Symposium on Robotics Research*, pp. 419–433, 2015.

[39] S. Bai, J. Wang, F. Chen, and B. Englot, "Information-theoretic exploration with bayesian optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1816–1822, 2016.

[40] S. Bai, F. Chen, and B. Englot, "Toward autonomous mapping and exploration for mobile robots through deep supervised learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2379–2384, 2017.

[41] 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.

[42] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, pp. 2829–2838, 2016.

[43] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl ^{2}: Fast reinforcement learning via slow reinforcement learning," *International Conference on Learning Representations (ICLR)*, 2017.

[44] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *International Conference on Learning Representations (ICLR)*, 2017.

[45] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 2759–2764, 2016.

[46] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," *arXiv preprint arXiv:1610.01733*, 2016.

[47] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 2140–2146, 2017.

[48] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364, 2017.

[49] S. Choudhury, A. Kapoor, G. Ranade, and D. Dey, "Learning to gather information via imitation," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 908–915, 2017.

[50] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2616–2625, 2017.

[51] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," *Advances in Neural Information Processing Systems*, vol. 29, pp. 2154–2162, 2016.

[52] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to Reinforcement Learning*, vol. 135. MIT Press, 1998.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[54] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[55] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *International Conference on Machine Learning*, pp. 1050–1059, 2016.

[56] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[57] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[58] "Random dungeon generator." `http://perplexingtech.weebly.com/random-dungeon-demo.html`.

[59] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning*, pp. 4470–4479, 2018.

[60] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[61] K. Chen, J. P. de Vicente, G. Sepulveda, F. Xia, A. Soto, M. Vázquez, and S. Savarese, "A behavioral approach to visual navigation with graph localization networks," in *Robotics: Science and Systems*, 2019.

[62] T. Wang, R. Liao, J. Ba, and S. Fidler, "Nervenet: Learning structured policy with graph neural networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[63] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction." Technical Report, Georgia Institute of Technology, GT-RIM-CP&R-2012-002, 2012.

[64] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2012.

[65] M. Kaess and F. Dellaert, "Covariance recovery from a square root information matrix for data association," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1198–1210, 2009.

[66] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[67] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[68] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[69] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "Segmatch: Segment based place recognition in 3d point clouds," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5266–5272, 2017.

[70] H. Gao and S. Ji, "Graph u-nets," in *International Conference on Machine Learning*, pp. 2083–2092, 2019.

[71] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241, 2015.

[72] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the International Conference on Machine Learning*, pp. 41–48, 2009.

[73] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, "Automated curriculum learning for neural networks," in *International Conference on Machine Learning*, pp. 1311–1320, 2017.

[74] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *International Conference on Machine Learning*, pp. 1515–1528, 2018.

Vita

Fanfei Chen

Education

PH.D. in Mechanical Engineering — January 2016 - August 2021
Stevens Institute of Technology, Hoboken, NJ

M.E. in Mechanical Engineering — August 2014 - December 2015
Stevens Institute of Technology, Hoboken, NJ

B.E. in Automobile Engineering — September 2010 - June 2014
Tongji Zhejiang College, Zhejiang, China

Publications

F. Chen, P. Szenher, Y. Huang, J. Wang, T. Shan, J. Wang, S. Bai, and B. Englot, " Zero-Shot Reinforcement Learning on Graphs for Autonomous Exploration Under Uncertainty," *International Conference on Robotics and Automation (ICRA),* June 2021.

S. Bai, T. Shan, **F. Chen**, L. Liu, and B. Englot, "Information-Driven Path Planning," *Current Robotics Reports,* vol. 2, pp. 177-188, April 2021.

T. Shan, J. Wang, **F. Chen**, P. Szenher, and B. Englot, "Simulation-based Lidar Super-resolution for Ground Vehicles," *Robotics and Autonomous Systems,* vol. 134, Article 103647, December 2020.

F. Chen, J. D. Martin, Y. Huang, J. Wang and B. Englot, "Autonomous Exploration Under Uncertainty via Deep Reinforcement Learning on Graphs," *Pro-*

ceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October 2020.

F. Chen, J. Wang, T. Shan, and B. Englot, "Autonomous Exploration Under Uncertainty via Graph Convolutional Networks," *Proceedings of the 19th International Symposium on Robotics Research (ISRR),* October 2019.

F. Chen, S. Bai, T. Shan, and B. Englot, "Self-Learning Exploration and Mapping for Mobile Robots via Deep Reinforcement Learning," *AIAA SciTech Forum,* pp. 0396, January 2019.

S. Bai, **F. Chen**, and B. Englot, "Toward Autonomous Mapping and Exploration for Mobile Robots through Deep Supervised Learning," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 2379-2384, September 2017.

S. Bai, J. Wang, **F. Chen**, and B. Englot, "Information-Theoretic Exploration with Bayesian Optimization," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems,* pp. 1816-1822, October 2016.