Benchmarking Deep Reinforcement Learning Methods for Decentralized Multi-Robot Exploration

by

Richard Ren

Supervisor: Goldie Nejat

April 2022

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Abstract

The traditional algorithms in multi-robot exploration fail to generalize to the diverse, limited communication environments found in urban search and rescue (USAR) missions. They employ heuristic functions or state features that are handcrafted for the specific terrain the methods were designed for. To alleviate these problems, the state-of-the-art use approaches from the field of Deep Reinforcement Learning (DRL). As a new field, it suffers from a lack of reproducibility. Many methods are published, but their performance can neither be verified nor reproduced in a different environment. From this gap, we developed a simulation environment representative of the terrain in USAR and ported a DRL method called DME-DRL for evaluation. We were able to reproduce its relative performance against the traditional method Nearest Frontier (NF). In the paper, DME-DRL outperformed NF in the number of time steps using their no-clutter 100% communication success environment. In our implementation, DME-DRL underperformed NF by 6% in the number of time steps in our cluttered 100% communication success environment with 10% probability of failed action and scan.

Acknowledgements

First, I would like to thank my supervisor, Goldie Nejat for providing me with the opportunity to work on the exciting research at the Autonomous Systems and Biomechatronics Laboratory. There, I was mentored by Aaron Tan, who provided invaluable guidance and helpful feedback which made this work possible. I would also like to thank the previous thesis student Federico Pizarro Bejarano who had paved the foundations for my project and implemented the traditional exploration methods.

Second, I would like to thank my parents, for their constant love, support and encouragement through my journey in Engineering Science.

Contents

1 Introduction

In Urban Search And Rescue (USAR) missions, human teams search for trapped victims in collapsed structures [1]. Within these buildings, the environment is unstructured, cluttered, and dangerous for humans to explore [2]. Explorers face hazards from broken wires, natural gas leaks, and the structure caving in. It is far safer and more efficient to employ a team of robots to traverse such terrains. However, it is a difficult problem to coordinate a search effort for multiple robots that maximizes explored area with respect to time. [3]. First, robots do not have a map of the damaged structure ahead of the rescue so they cannot plan paths in advance [2]. Second, during exploration, a robot may not know the regions its team members are exploring or have already explored due to limited communication in the damaged building [1]. While physical obstructions such as walls and rubble or electrical interference may impede communication, robot team members are also unreliable. They may be damaged in a further collapse or their mechanisms could fail.

Traditionally, Multi-Robot Exploration (MRE) methods were based on crafted heuristics that made strong assumptions about the optimal location that will gain the most information about the environment[2]. As an example, [4] assumes exploring corridors are more desirable because they connect to unexplored rooms on a floor.

The current state-of-art uses approaches from the field of Deep Reinforcement Learning (DRL) [2]. In Reinforcement Learning (RL), goal-directed agents attempt to learn a behavior that maximizes a reward signal through interactions with its environment [5]. In the context of MRE, robots learn how to make the best decision through their experiences of exploring different environments, where they are rewarded for efficient exploration decisions and punished for inefficient ones. However, the applications of RL are limited to problems with environments that have a small state space like board games because of past computational constraints. Researchers may also reduce the state space by handcrafting features to encode the state. However, MRE involves environments that have a very large state space. With the advent of deep learning and the increase in computational power, DRL uses neural networks with millions of parameters to encode high-dimension observations of the environment from robot sensors into low-dimension feature representations and to decide where each robot should explore next. These parameters are incrementally updated through the interactions with the environment to develop the desired behavior.

Since DRL is a popular and novel field, many papers are published without a means to verify results. For instance, authors may omit their implementation or details about the training hyperparameters or the network architecture that are needed to replicate the experiments. Even if their implementation is included, there may be errors in the code or their implementation may not achieve the desired performance. Furthermore, when researchers attempt to test the author's methods in different environments, they may find the techniques fail to generalize [6]. Driven by this need for reproducibility, we seek to develop a USAR simulation environment and then implement a benchmark for MRE. We will examine the literature on traditional and state-of-the-art methods and implement the algorithm that satisfies our criteria.

2 Literature Review

The multi-robot exploration problem is defined as: given a team of robots, each possessing a local map and sensors to observe their environment, the objective is to find the path which minimizes the exploration time or the distance travelled for each robot, while fully exploring their environment [7]. At each time step of the episode, robots explore the environment by scanning the environment, and if in the proximity of a teammate, communicate information [7]. With these observations, the algorithm must decide a location or direction for each robot to head towards [7]. Typically, these targets are located on the frontier of the map, which is the edge between the explored and unexplored area [2].

Existing multi-robot exploration methodologies can be categorized as 1) centralized [8] and 2) decentralized [8], where the former incorporates a global coordinator which uses all of the robots' observations to decide the target for each robot, and the latter allows each robot to use their own observations to select a target. There are also hybrid architectures which divide the team into subteams and make decisions using designated subteam coordinators[8]. Once a termination condition based on the team's total exploration has been reached, then the episode has completed.

In the remainder of this section, we review several approaches to MRE to select an algorithm for implementation. We will first describe traditional methods and then proceed to the state-of-the-art.

2.1 Traditional Methods

Traditional MRE algorithms focus on crafting a heuristic to decide where each robot should explore. They can be further divided into Utility-based [4, 9, 10] , Market-based [11, 12, 13], and Planning [14] methods .

2.1.1 Utility-Based Methods

Utility-based methods include the design of a team utility function and a centralized process to match robots with targets that maximize the team utility. There are three main ideas in utility-based methods. First, they seek to maximize the total information gain from the unknown environment when assigning a robot to a particular target. [9] defines information gain as the increase in the knowledge of the environment, or the reduction of entropy, where entropy represents the uncertainty in the exploration potential for a particular cell. For instance, [4] assumes corridors have higher semantic information because they can connect to unexplored rooms on a floor, and incorporate this assumption into their utility function. [4] tested the performance of their utility function with teams of 5-50 robots in 2D simulations of floor plans in real buildings such as the Intel Research Labs and Fort Sam Huston hospital as shown in Figure 1. In Figure 2, by incorporating semantic information from corridors, they significantly reduced the exploration time.

Second, there is a distance cost for the robot to travel to the target [4, 9, 10]. Third, there is a factor which discourages robots from exploring the areas near other members. As an example, [9] greedily matches the robot to the target with the greatest utility and discounts the remaining targets that are close to those already selected. [9] tested their greedy discounting method in their physical re-

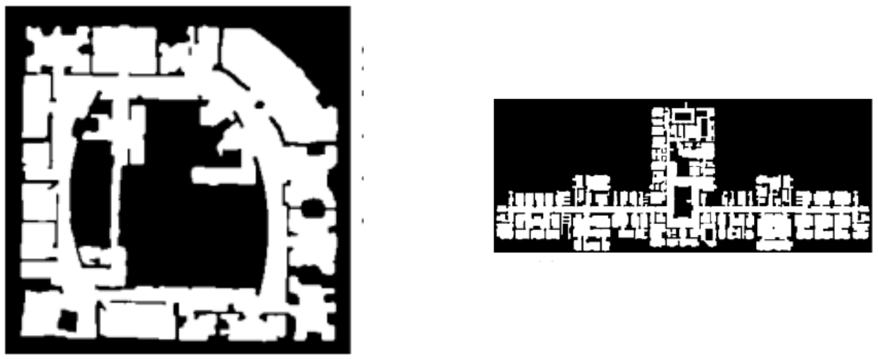


Figure 1: Floor plans of Intel Research Lab (left) and Fort Sam Huston Hospital (right) [4].

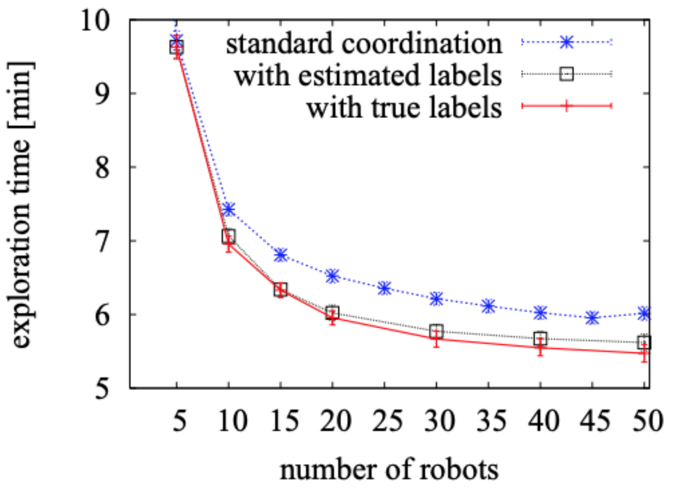


Figure 2: Comparison of results for semantic information utility function (with estimated/true labels) against no semantic information (standard coordination) [4].

search lab as well as in two 2D simulation environments, one simple and one complex (Figure 3). In the simulation, they compared against Nearest Frontier Exploration, MinPos, and Entropy methods (Figure 4) . The Nearest Frontier Exploration (NFE) is a common benchmark, in which, each robot on the team greedily selects the frontier cell for exploration that is closest to them at that time step, regardless if other robots select the same or a nearby cell [15]. MinPos extends NFE, by allowing a robot to select a frontier cell only if it is the closest robot to the cell [16]. Finally, the Entropy method uses a utility function that only considers information gain alone [9].

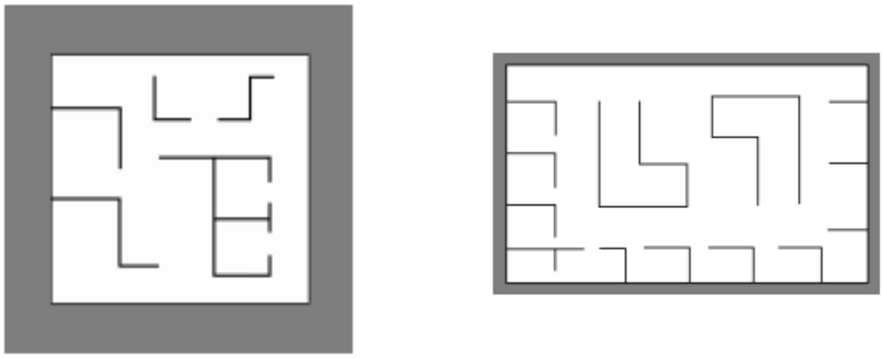


Figure 3: Floor plans of simple (left) and complex (right) 2D test environment [9]

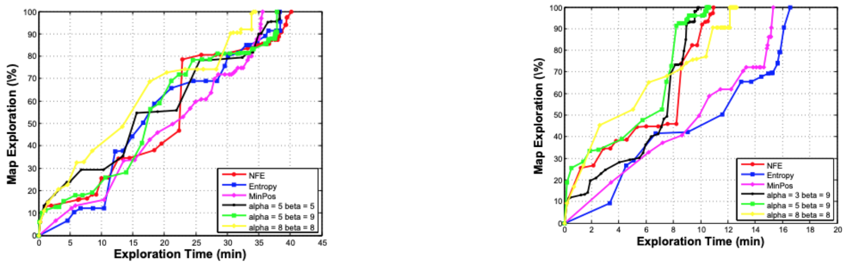


Figure 4: Comparison of results for simple (left) and complex environment (right). [9]'s proposed coordination methods are labelled with alpha and beta, while NFE, MinPos, and Entropy are the benchmarks [9].

2.1.2 Market-Based Methods

While market-based approaches also employ the concept of utility functions, their novelty comes from considering the robot team as a market, in which members trade tasks and resources with one another to maximize individual utility functions in a manner which also maximizes the team utility function. To achieve this balance, markets use a single, centralized or multiple subteam auctions which receive teammates' bids for tasks and resources, then distribute the targets in a way that maximizes team utility [11]. Like utility methods, [12] uses a central executive when assigning robots to targets from the bids they have submitted. In this paper, a bid is a list of (target, utility) pairs that robots compute independently using their local observations. [12] tested their method with real robots in Fort Sam Huston hospital, but published results based on their 2D simulations. They did not benchmark against other methods, but measured the amount of time it took for robots to cover a range of percentage of their environment (50, 90, 95, 100) while varying the number of robots on the team from 1-3.

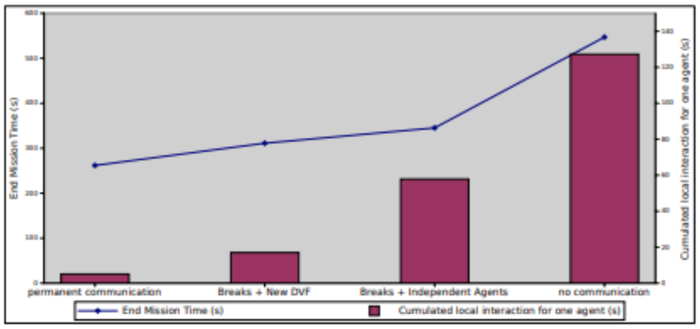
However, [13] draws more ideas from markets. First, robots generate a list of targets to explore, either randomly, greedily, or by recursively dividing the unknown regions using quadtrees and picking the centres of those regions. Robots submit their targets in order of increasing utility to the central executive, and the team members bid for the right to explore the target. They submit a priced bid if the target is profitable, in other words, the information gained is greater than the cost to travel to the target. The highest bidder wins the right to explore the target. [13] tested their market architecture in 3 different real world environments using a team of 4 robots. The first environment was located indoors in the Field Robotics Center highbay which is a large, cluttered space that contains furniture and other robots. The second was an outdoor patio with a mix of open areas and some walls and tables. The third environment was a hotel conference room that was filled with tablees and people walking around.[13] defined their performance metric as a quality of exploration Q which is found by dividing the area covered by the distance travelled by all the robots. Under the market architecture and using random and quadtree goal generation they were able to achieve a Q of $1.4m^2/m$, while greedy goal generation produced a Q of $0.85m^2/m$. They benchmarked their results against robots individually generating goals randomly and exploring which attained a Q of 0.41.

2.1.3 Planning-Based Methods

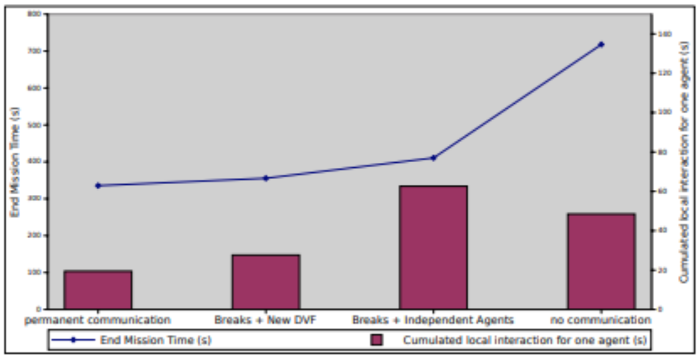
Planning-Based methods view MRE as a type of Markov Decision Process (MDP) problem. They seek to determine a policy which maps the robots' states to probabilities of selecting possible actions at each time step [5]. To improve the policy, planning methods optimize for a value function [5]. Value functions represent the total expected return of future rewards for being in a particular state [5]. Rewards are given at the end of each time step based on the state with respect to the goal of exploration [5].

In particular, [14] views MRE as a Decentralized Partially Observable Markov Decision Problem (Dec-POMDP). In a Dec-POMDP, robots do not have access to the state information, instead they can only partially observe the state due to the physical limitations of their sensors and communication [17]. Robots use these observations to maintain a probability distribution over states. They seek to find the optimal policy by selecting actions to maximize the value function. [14] proposes modifications to Distributed Value Functions (DVF) to solve for the Dec-

POMDP and find the optimal policy. Their modifications accommodate for limited communication and approximations to state transition probabilities for faster computation. Similar to the methods mentioned above, they tested in both real and 2D simulated office-like environments, but only published numerical results for the simulations. Instead of benchmarking against common methods, [14] chose to test different communication schemes and ways to solve the Dec-POMDP. As shown in Figure 5, permanent communication uses DVFs and achieves the best result. In experiments that consider communication breaks, the robots either solve the value functions independently or rely on the modified DVF algorithm. Finally, in the independent, no communication case robots solve the policy using value iteration.



(a) office-like1 with 4 robots.



(b) office-like2 with 3 robots.

Figure 5: Exploration Time for [14]'s method

2.2 DRL Methods

Like Planning Methods, DRL methods seek to learn a policy that optimizes an objective formulated from efficient exploration and use MDPs for problem formulation. Planning methods use dynamic programming (DP) to solve the MDP, because they define a perfect model of the environment using handcrafted features [5]. However, since USAR environments are dynamic and have a large state space that cannot be encoded into static, handcrafted features, DRL seeks to develop a model of the environment using learning [2]. In DRL methods, the policy is a neural network. At the end of each time step, the policy is given a reward based on the current state determined by a reward function. All functions reward based on the size of the new area that is explored and punish wasted exploration time [7, 18, 19, 20]. As examples of the latter idea, [7, 20] give a penalty every time step to encourage the robot to speed up the exploration. Both [7, 20] tested their idea against the Nearest Frontier Exploration Method as a benchmark with a team of robots in 2D indoor simulations. By varying the number of robots on the team from 1-4, [20] reduced the average moving distance per robot by 34-32% in maze environments and 47-32% in office-like environments. The environments in DME-DRL were less cluttered, and demonstrated an improvement of 3%.

On the other hand, [18, 19] penalize by accounting for robots retracing explored areas. [18] tested on a 20x20 grid world with a team of 12 robots. For a set 65 time steps, the team explored 92.6% of the environment with their proposed method in comparison the Nearest Frontier which achieved 90.4% averaged over 1000 random maps. As an extension of their method [19] tested in the same environment with 10 robots over 1000 random maps for 30s. Their new method explored 95.2% of the environment in comparison with their old method which covered 89.2%. Both [19, 20] also give a larger, terminal reward that depends on whether the teams had fully explored the environment at the end of the episode.

Tuples consisting of the observation, action, and the reward, are used to train the policy under a modified state-of-the-art DRL multi-agent learning paradigm. [7] uses a modified version of MADDPG [21], while [18] uses the CommNet architecture [22] and then adds an attention mechanism to produce [19]. These learning paradigms are categorized as policy gradient methods. These methods update the parameters of the network in the direction of the gradient to maximize the total expected reward from the entire exploration episode.

2.3 Benchmark Method

2.3.1 Selection

Recall USAR missions occur in collapsed structures, in such uncertain environments, team members could either be obstructed by physical structures like walls and rubble or be damaged by sudden impact from a fall or debris [1]. Because of these conditions we desire a decentralized algorithm, since it can operate under conditions of limited communication between robots and robot failures. That way robots are able to decide the target that will maximize the team's explored area with only their local information, unless they can come in proximity with another robot and share information with each other.

We cannot use centralized utility methods that achieve published results when there is a global coordinator on the team. Global coordinators require constant communication to receive robot position and observations to select targets. Similarly hybridized market-based methods are infeasible since a subteam of robots may not be close together to form an auction. While planning methods are decentralized, they handcraft robot state features and suffer from the curse of dimensionality when dealing with high-dimensional features [2]. The curse of dimensionality states that high-dimensional features require an exponential increase in the number of observations for accurate function approximation [23]. However, each robot has a limited number of observations in each time step. Furthermore, utility, market, and planning methods involve the design of functions that rely on tuning hyperparameters sensitive to different environment conditions [2]. Recall that [4] included a semantic information factor encouraging the robots to explore corridors. However, to achieve better performance in environments with few corridors, it may be advantageous to reduce the weight of semantic information. Thus, decentralized DRL methods are best suited for our use case. While [18, 19] classified their DRL algorithm as decentralized, the CommNet architecture requires a global communication channel that can be accessed by all robots at each time step [22]. In addition, [20] is a DRL method that merges all of the robot's observations to decide where each of the robots should go for each time step. Therefore, we have selected the paper "Decentralized exploration of a structured environment based on multi-agent deep reinforcement learning" (DME-DRL) as our benchmark [7].

2.3.2 Network Architecture

In DME-DRL, the goal is finding a path that minimizes travel distance and fully explores the environment. It applies the MADDPG algorithm to learn the policy. Each robot has an actor network which acts the policy and decides the direction to move at each time step as well as the critic network which estimates the value of the policy. Below, Figure 6 shows the architectures for the actor and critic network.

The inputs into the critic and actor networks are the robot's observations which consist of their grid map and a position vector with the positions of the other robots that it sees at the time step. The actor network actually stacks the observations for the past 6 time steps. Since the grid map can be thought of as an image, the convolution and ReLU layer extracts features from the map [24]. Then the features

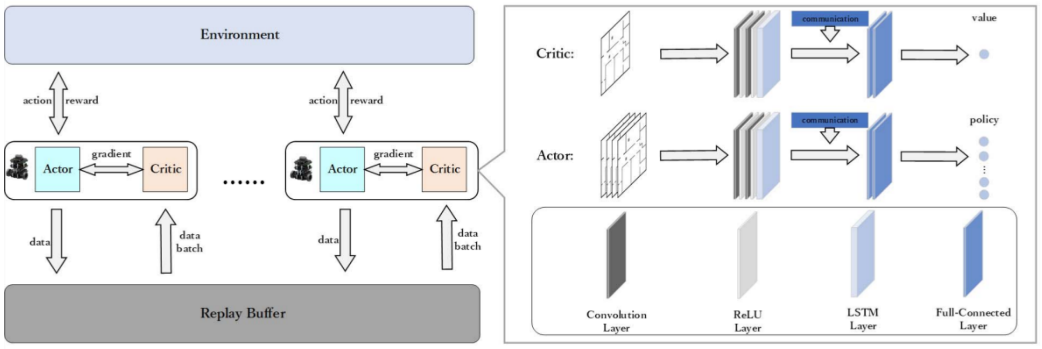


Figure 6: Network architectures for actor and critic models [7].

are augmented by a LSTM layer which considers the history of observations [25]. Finally, the augmented features are passed through the fully connected layer to produce a value for the critic and a probability vector of actions for the actor.

2.3.3 MADDPG

Multi-agent deep deterministic policy gradient (MADDPG) is a DRL algorithm used in multi-agent problems.

Previous approaches to multi-agent domain include Q-learning [26], in which agents attempt to learn their own optimal action-value function Q^* which measures their policy. This is achieved by minimizing the loss between a $Q*'$ which is a copy of Q that is updated periodically.

$$L(\theta) = E_{s,a,r,s'}[(Q^*(s,a|\theta) - y)^2]$$

$$where\ y = r + \gamma max_{a'} Q^{*\prime}(s', a')$$

However, because agents are learning independently, the environment appears non-stationary to each of the policies and learning cannot converge [21].

Another approach to the multi-agent problem are Policy Gradient methods [27], which seeks to optimize the parameters θ of the policy π to maximize an objective function J by making updates in the direction of the $\nabla_\theta J(\theta)$.

The gradient is often in the form of:

$$\nabla_\theta J(\theta) = E_{s,a}[\nabla_\theta log \pi_\theta(a|s) Q(s,a)]$$

But in multi-agent settings, an agent's reward depends on the actions of other agents and this results in high variance in the gradients [21]. Deep Deterministic Policy Gradients [28] are a subset of policy gradient methods that asserts the policy μ always chooses the same action given a state rather than produce a probability distribution based on a state. As a result the gradient becomes:

$$\nabla_\theta J(\theta) = E_s[\nabla_\theta \mu_\theta(a|s) \nabla_a Q(s,a)|_{a=\mu(s)}]$$

MADDPG extends DDPG and seeks to remedy the high variance and non-stationarity problems associated with the multi-agent domain. It relies on the centralized training decentralized execution (CTDE) and actor-critic framework [21]. Each robot has an actor that decides the actions at each time step with local information and a critic network which evaluates the actor's policy. During training, the critic network is granted access to global information such as the agent's policies, actions, and observations. The process of training consists of two parts.

Similar to Q-Learning, the critic is updated by minimizing a loss between the depends on a critic value which receives the global state information x like observations and a critic value which receives global state information x' at the next time step with the exception that the actions have been produced with target policies using the observations at the current time step [21].

$$L(\theta_i) = E_{x,a,r,x'}[(Q_i^{\mu}(x, a_1, ..., a_n) - y)^2]$$

$$where\ y = r_i + \gamma Q_i^{\mu'}(x', a_1', ..., a_n')|_{a_j'=\mu_j'(o_j)}$$

Like policy gradient methods, the parameters of the policy are updated in the direction that maximizes the critic's output [21].

$$\nabla_{\theta} J(\mu_i) = E_{x,a}[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{a_i}Q_i^{\mu}(x, a_1, ..., a_n)|_{a_i=\mu_i(o_i)}]$$

2.3.4 Training and Evaluation

To train this network, robots must gather sufficient exploration experience through random exploration for their replay buffer. An experience is a tuple consisting of the observation o at the current time step, action a, reward r, and observation $o^{'}$ at the next time step. The action a is the direction that is selected by actor network, and the robot routes a path to the closest frontier target in that direction using the A* algorithm [29]. The reward r is calculated as follows:

$$r_i^t = w_1 c_i^t + w_2 d_i^t, \tag{1}$$

where:

- r_i^t is the reward for robot i at time step t
- $w_1 c_i^t$ is the product between a scalar w_1 and c_i^t is the area explored in time step t.

- $w_2 d_i^t$ is the product between a scalar w_2 and d_i^t is the distance travelled in time step t.

Once the robots have explored a set number of episodes so that the replay buffer has sufficient experience, robots will continue to gain experience but also update the weights for training at a set number of time steps. Training consists of sampling a batch j from the replay buffer and the critic estimates the value of the robot's state in that experience.

$$y^j = r_i^j + \gamma Q_i^{'}(o^{'j}, a_{1:N}^{'})|_{a_k^{\prime}=\mu_k^{\prime}(o_k^j)}$$

where:

- y^j is the estimated value of robot i's state
- r_i^j is the reward of robot i at that sampled time step from experience tuple j
- γ is the discount factor
- $Q_i^{'}(o^{'j}, a_{1:N}^{'})|_{a_k^{\prime}=\mu_k^{\prime}(o_k^j)}$ is the total estimated future rewards. Note that $Q_i^{'}$ is a copy of the critic network and the inputs are $o^{'j}$ the observation at the next time step and actions $a_{1:N}^{'}$ are obtained by passing in the observation at the current time step o_k^j into a copy of the actor network $\mu_k^{'}$.

This value is used in minimizing the following loss function to update the weights of the critic.

$$L(\theta_i^Q) = \frac{1}{N}\sum_j (y^j - Q_i(o^j, a_{1:N}^j))^2 \qquad (2)$$

where:

- $L(\theta_i^Q)$ is the critic value loss.
- y^j is the estimated value of robot i's state based on the Bellman Equation [5].
- $Q_i(o^j, a_{1:N}^j)$ is the estimated value of robot i's state using the critic network, whose inputs are the observations and actions at the current time step.

The weights of the actor move in the direction of the policy gradient to maximize the value.

$$\nabla_{\theta_i^\mu} J \approx \sum_j \nabla_{\theta_i^\mu} \mu_i(o_i^j) \nabla_{a_i} Q_i(o_{1:N}^j, a_{1:N}) \tag{3}$$

where:

- $\nabla_{\theta_i^\mu} J$ is the gradient of the cost function for the actor
- $\nabla_{\theta_i^\mu} \mu_i(o_i^j)$ is the gradient of action with respect to the actor network parameters
- $\nabla_{a_i} Q_i(o_{1:N}^j, a_{1:N}))$ is the gradient of the value of the robot's state with respect to the action

Note that during evaluation, robots explore an environment with the actor choosing an action at each time step and never use the critic.

3 Methods

As described in the introduction, the scope of this thesis is to develop a USAR simulation environment and then attempt to reproduce the performance of our selected method: DME-DRL. In this section we will first describe the design of our simulation before diving into the modifications we made in the DME-DRL algorithm.

3.1 Simulation Design

3.1.1 Environment

Originally DME-DRL was trained on 100 maps and tested on the HouseExpo dataset, which is a collection of 2D indoor floor plans [30]. DME-DRL selected 100 floor plans from HouseExpo and resizes them to a 200x200 array.

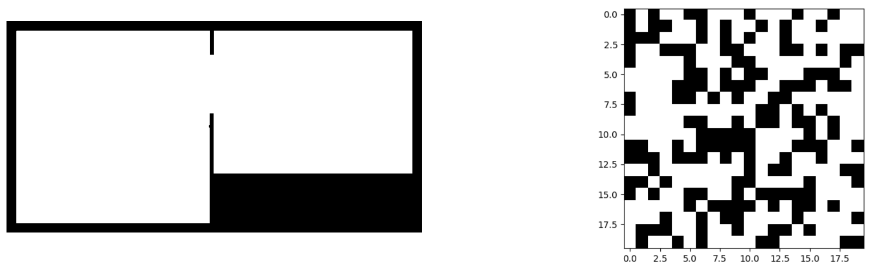


Figure 7: A HouseExpo floor plan (left), compared with a randomly generated grid map (right). Note that white indicates free space and black indicates obstacles.

In order to better simulate the clutter in a USAR environment, we trained and tested on 20x20 randomly generated grid worlds where the obstacle density is randomly chosen between 30% to 40% so it experiences a variety of cluttered scenes. To follow the original training scheme, we trained on a set of 100 randomly generated maps. The obstacle locations are also randomly generated to create an unstructured space. In this cluttered space exploration becomes much more challenging, with a greater number of obstacles that can represent debris or holes in the floor, robots can end up in far more dead ends and might discover it is not possible to fully explore the environment. With fewer possible paths robots are inclined to run into one another and retrace each other's exploration. Hence, high clutter environments also test the algorithm's effectiveness in dispersing the robots.

3.2 Robot Design

3.2.1 Scan Protocol

The robots in DME-DRL used a laser sensor with a circular range of 40 to scan the environment for robots, obstacles, and free space at each time step. The methods proposed by our lab involve robots with a laser sensor with a grid range of 4 instead. A robot can only see a cell within its range so long as there is no obstacle located along the line of sight calculated by the Bresenham algorithm [31].

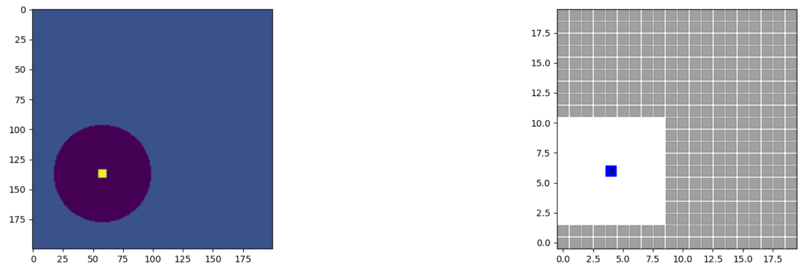


Figure 8: Range for DME-DRL Robot Sensor (left), compared with range for Benchmark Robot Sensor (right). Note that blue (left) and gray (right) cells represent unknown cells, while purple (left) and white (right) cells represent free space. The yellow square on the left is the robot and the blue square on the right is the robot.

3.2.2 Communication Protocol

DME-DRL experiments with three communication protocols between robots:

1. Complete Communication: robots share their maps with each other if they are within the broad synchronization range.
2. Layered Communication: robots record each other's positions if they are within the broad synchronization range and share their maps if they are within the smaller communication range.
3. No Communication: robots do not communicate any information with each other.

Our lab is interested in a communication protocol with a single communication range. Robots share their maps and record each others' positions if they are within the communication range.

After our modifications to the scanning and communication protocol, we replaced their edge detection method to detect frontier cells and instead mark free

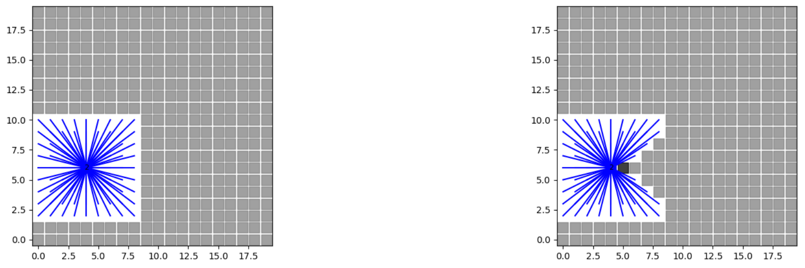
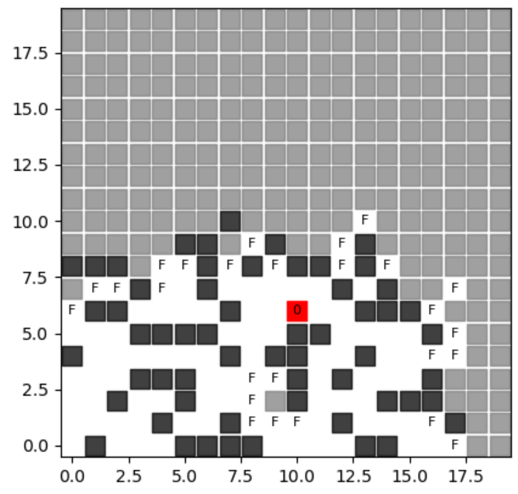


Figure 9: Range for Benchmark Robot Sensor without obstacles (left), compared with range for Benchmark Robot Sensor with obstacles (right). The blue lines indicate the lines of sight from the robot.

cells that neighbour at least one unknown cell as frontier cells. When merging maps together, we check if each of the cells in each of the frontier fulfills the above condition.



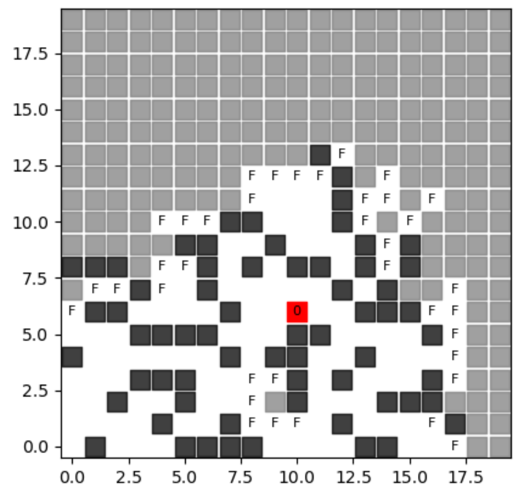


Figure 10: Robot map before communication (top), compared with robot map after communication (bottom). Note that the black cells indicate obstacles, the gray cells unexplored area, and white cells are free space. Free cells with F are also frontier cells, which mean that they neighbour an unexplored cell. The robot is demarcated in red with the id of 0.

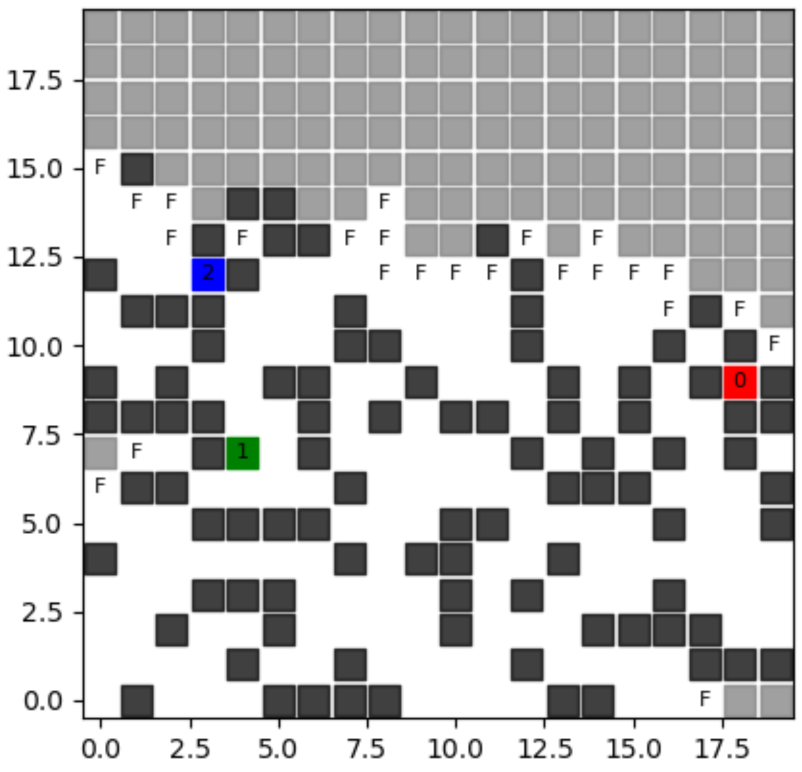


Figure 11: Visualization of the merged team robot map.

3.3 Modifications to DME-DRL Algorithm

The authors of DME-DRL have published their implementation on Github [32]. However, we have made several modifications to benchmark against the methods proposed by our lab. In addition, we have developed visualizations to verify the logic of the modifications.

3.3.1 Network Architecture

We reduced the number of weights in the LSTM layer of the actor and critic networks since the robot maps decreased in size from 200x200 to 20x20.

3.3.2 Training Hyperparameters

We increased the magnitude of the hyperparameters in the reward function while maintaining the reward ratio. We set w1 = 0.02 and w2 = 1. We discovered that this decrease the number of steps by 10% during training. In addition, we set the heuristic to the Euclidean distance in their A* implementation to be consistent with our lab's pathfinding methods. Other than these changes, the hyperparameters match those of the DME-DRL implementation.

3.3.3 Communication Protocol

In the original algorithm, each robot moves to its destination selected at the beginning of each time step one at a time as shown in Figure 12.

The robot does not communicate with others until it has reached its destination. This means that each robot is not using the latest information to make a decision on where to go even if it has been communicated to them. As a result, in

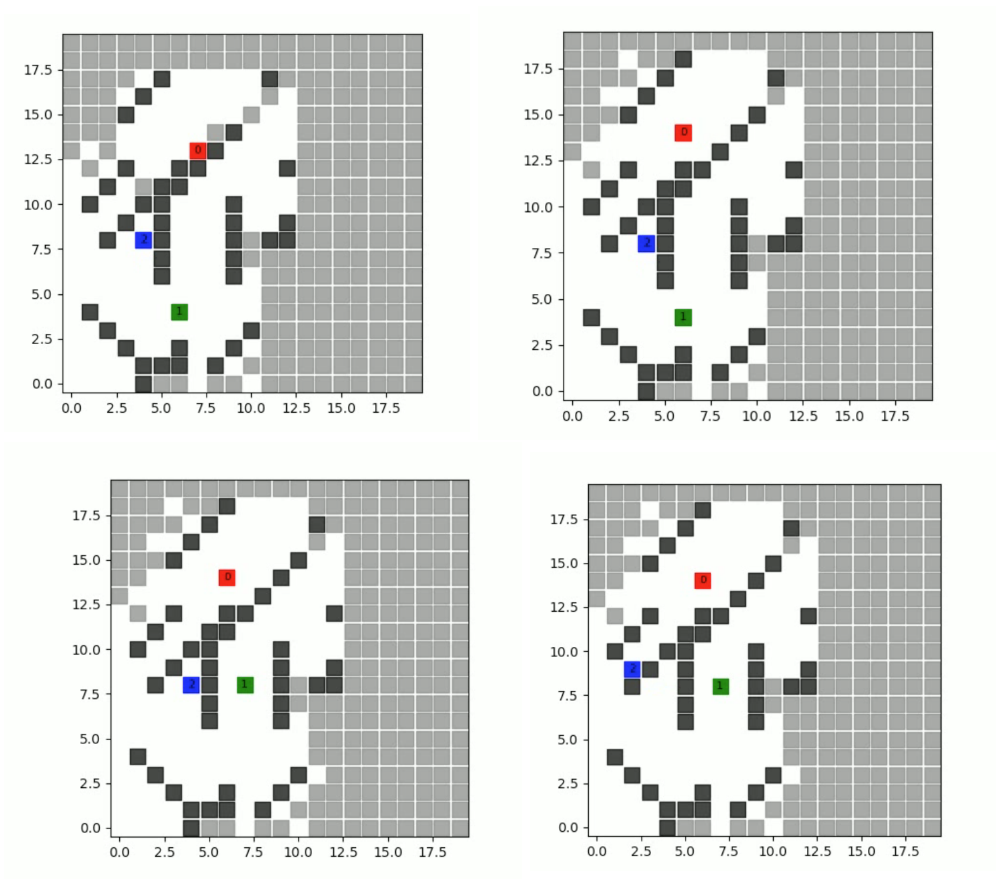


Figure 12: Illustration of the sequential order in DME-DRL. The robots begin the time step in the top left image. First robot 0 moves to its destination (top right), second moves robot 1 (bottom left), and third moves robot 2 (bottom right). When robot 2 has reached its destination, the time step has completed.

the original implementation robots use stale information to plan a path that could traverse through recently explored territory.

Our solution to this problem is stopping the robot that is in motion when it comes into the communication range of one or more robots. For example, if robot 0 comes into the range of robot 1, it stops and the two robots communicate and merge maps. Next, robot 1 feeds these new observations into its actor network and selects a new destination along the frontier. Robot 1 will move towards its destination unless it comes into communication range of other team members it has already seen i.e. robot 0. The set of robots that is has seen is cleared after all the robots have reached their chosen frontier cell.

With this modification, we ensure each robot decides where to go using the latest information. In addition, stopping the robots when they come into communication range will also count the number of local interactions or the times when the robots come into communication range more accurately.

3.4 Testing

We generated 10 20x20 grid world test environments with obstacle density of 30-40%. All of these environments are fully explorable. We measure the performance of DME-DRL's exploration against the following metrics:

1. Total steps: We count the total time steps that the robots have taken for exploration. In a time step, each team member has moved one cell.

2. Total Travel Distance: We measure the total distance that is travelled by the robot team. If a robot moves one cell in the cardinal directions, that corresponds to a distance of 1. However, if a robot moves one cell diagonally that corresponds to a distance of $\sqrt{2}$.

3. Number of Local Interactions: We count the number of times that one or more robots come within communication range of each other.

4. Objective Function Value: We measure the efficiency of the team's explo-

ration with respect to a custom metric:

$$\sum_{j=1}^{h} D_j^{-1}|E_j|, \tag{4}$$

where:

- h is the total number of steps.
- D_j^-1 is the inverse of the joint distance travelled for time step j
- $|E_j|$ is the percentage of total area explored for time step j. Using percentages allow us to compare the objective function value in different environments because we are normalizing the range.

Since the objective function value may vary due to the number of steps in an episode, we interpolate the total area explored and the joint distance travelled to 9000 data points. This number gives fine grained plots for percentage of total area explored vs joint distance travelled.

We compared these metrics against implementations of the traditional methods we have described in the literature review: Nearest Frontier, Utility-Based, and Planning Based by a previous thesis student. All methods used a team of 3 robots. These methods were not designed with considerations of robot unreliability. As mentioned in the introduction, robot team members can fail. We introduced the following probabilities of failure:

- Scan: a robot has a 10% chance to miss recording a cell in their field of view.

- Communication: a robot may fail to communicate with its team member. If it fails, each robot is unable to communicate for the next 7 steps. We tested in environments with 0, 20, 50, 100% of communication failure.

- Action: a robot has a 10% chance to fail at its intended action. If it fails, then a robot randomly picks a free neighbouring cell and moves there.

To make a fair comparison with traditional methods, we did not train DME-DRL with failure probabilities. However, we subjected all methods to the probabilities of the failure in the test environments.

4 Results and Discussion

In this section we will present the training plots for our final model as well as the results that our modified DME-DRL achieved in the test environment.

4.1 Training

As expected in Figure 13, we see the 3 critic losses converge as the training proceeds, which means the critic becomes more accurate in judging the value of the policy. Interestingly, the actor loss, which is the negated critic value, also converges as time progresses, so the actor produces actions which have the same value according to the critic.

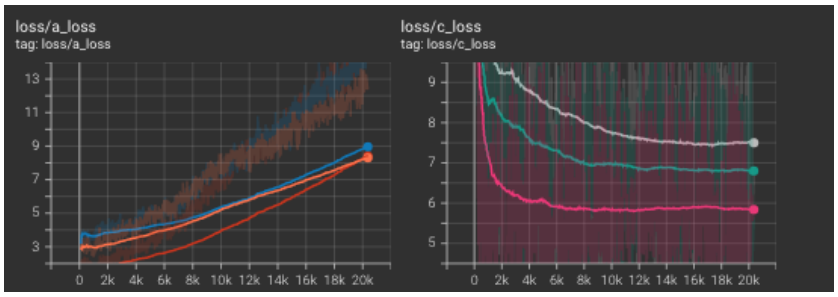


Figure 13: Loss plot for actor (left) and critic (right, see equation 2) networks during training. We used three robots to explore the environment, hence we track three critic and actor losses each.

While we expect the mean and individual rewards to increase as it explores more maps, we see in Figure 14 while we expect the mean and individual rewards to increase, it shortly plateaus after recovering from an initial drop. This suggests that the training process has not improved a robot's ability to explore efficiently. In other words, the robot's selection of a target is slightly better than random. It is important to note that in reinforcement learning, the reward plots take precedence over the loss plots.

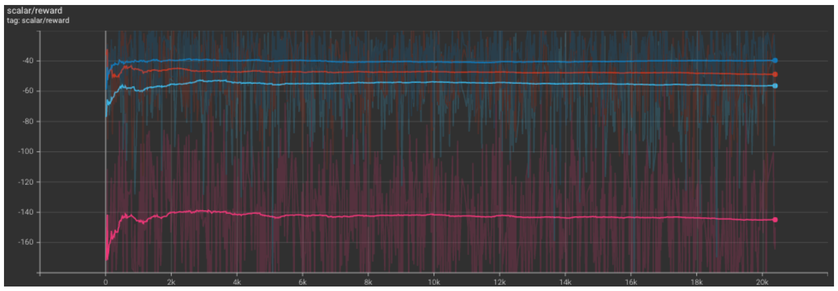


Figure 14: Team mean reward (left) and individual reward (right) (see equation 1) plot during training. We used three robots to explore the environment, hence we track the rewards of the three robots in the individual reward plot.

To get a better idea of the learning process of the robot during training we also recorded other metrics for debugging purposes. For efficient exploration, we are interested in how many time steps the robots take to complete a training episode. The range in the number of steps is about 3 steps. If the algorithm was learning effectively, we would expect the distance to decrease, but this behavior is not produced in Figure 15. This may be a fault with the network architecture.

Figure 16 shows the progress of team over several time steps in a training exploration episode.

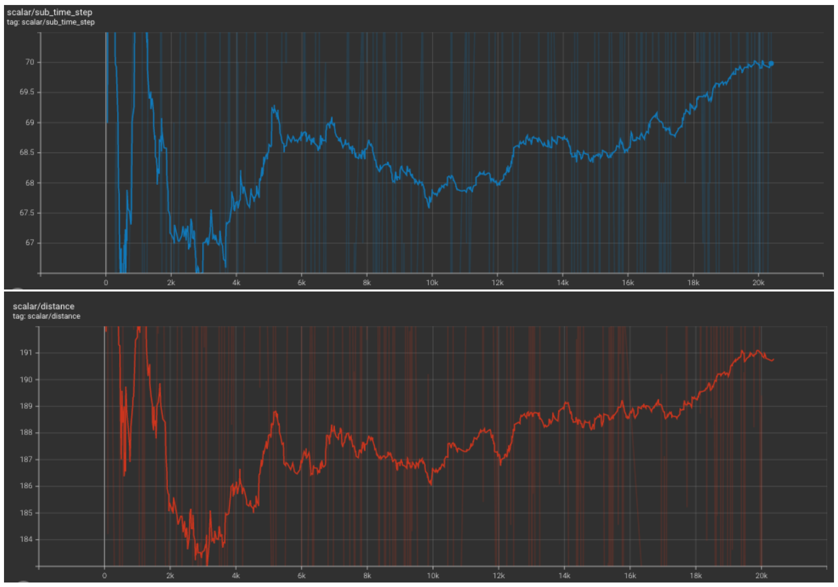


Figure 15: Number of steps (top) and joint distance travelled (bottom) during training.

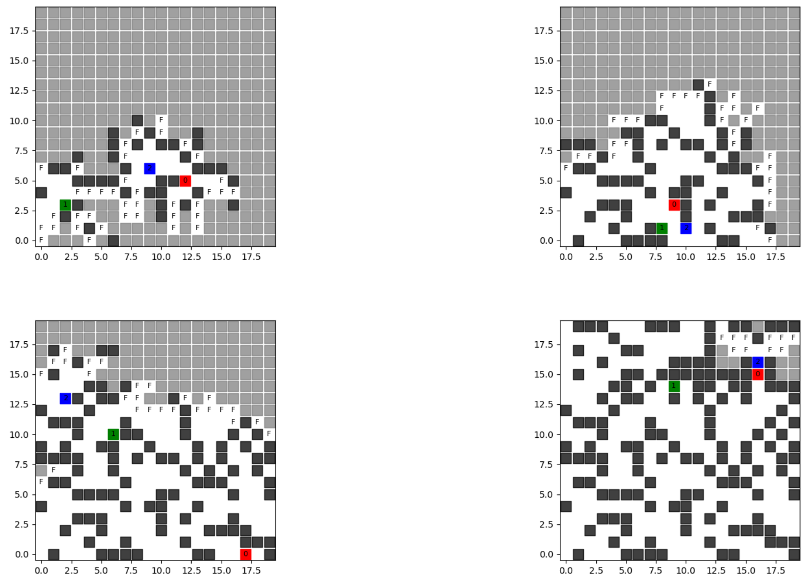


Figure 16: Progress of team exploration over at four different time steps. The chronological order of steps is from left to right, and top to bottom. Observe how the team diffuses throughout the environment to explore different parts of the map.

4.2 Test

We tested in 10 different 20x20 randomly generated grid worlds with an obstacle density of 30-40%. For each map we tested with 4 different probabilities of communication success: 0, 50, 80, 100. For each map and probability we started a team of 3 robots in the four different corners of the environment.

These environments were fully explorable, meaning that all cells could be scanned by the team. As mentioned before in 3.4 we introduce the probability of failure in the test environments. We included the results from Nearest Frontier, Utility-Based, and Planning Based to provide a benchmark for DME-DRL.

4.2.1 Steps and Distance

Figure 17 shows the number of time steps taken and the team's distance travelled by each of the methods in the test environment averaged over each probability of communication success. The number of steps has increased from 70 during training to 81. This significant increase comes from the unreliability noise and a weakness of DME-DRL. The authors state that it performs poorly when we start the robots close together as in Figure 18 [7]. In the other communication schemes it performs comparably with traditional methods.

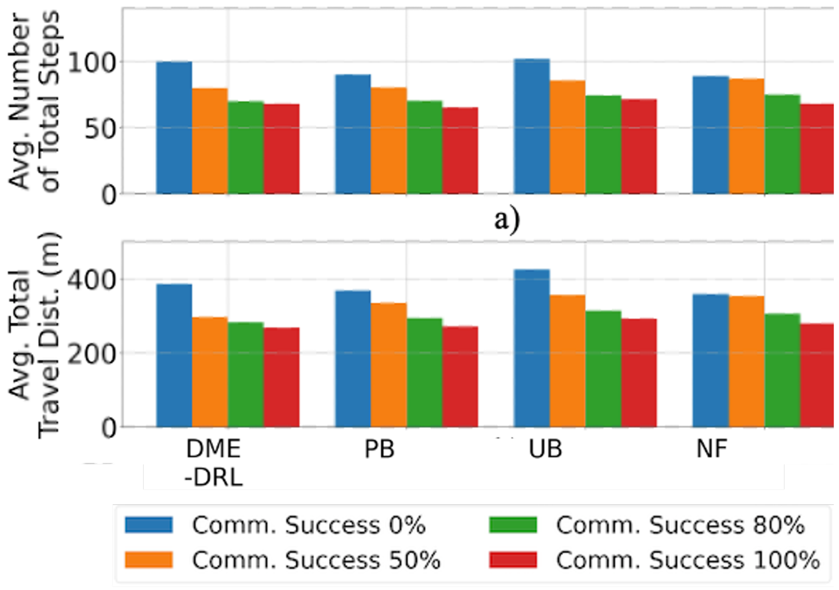


Figure 17: Average Number of Total Steps and Average Total Distance in Test Environment

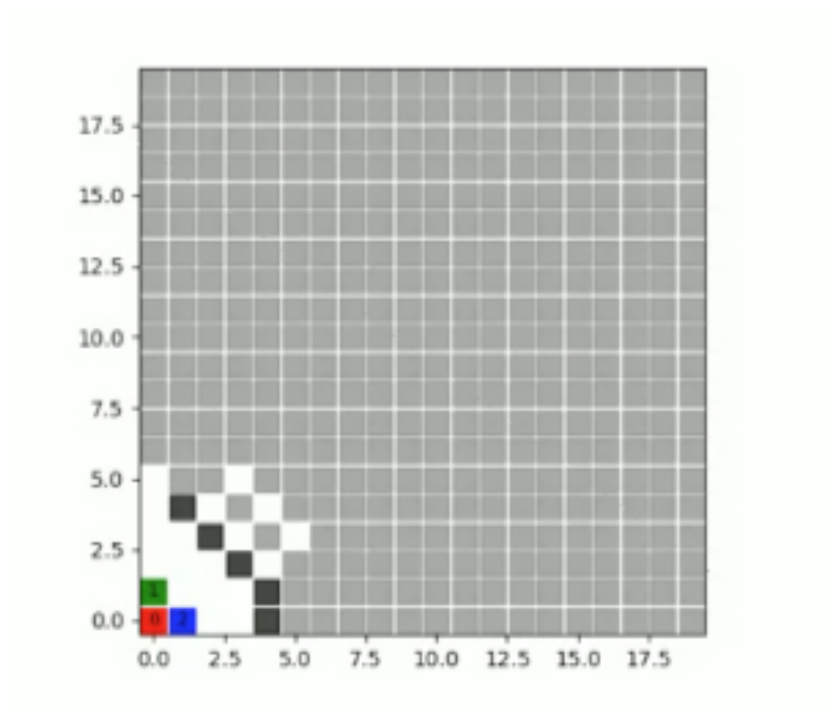


Figure 18: Close initial positions in test environment induce poor performance in DME-DRL.

DME-DRL was published to outperform Nearest Frontier by 3% in the number of steps in the 100% communication case and without probabilities of failure. In our tests, DME-DRL underperformed Nearest Frontier by 6%. Thus DME-DRL's performance in the test environment was slightly degraded from Figure 19. This could have occurred for several reasons:

- The authors do not include failure in action, communication, or scan.
- The authors trained DME-DRL in environments without clutter.
- The test environment starts robots close together which exploits their weakness.

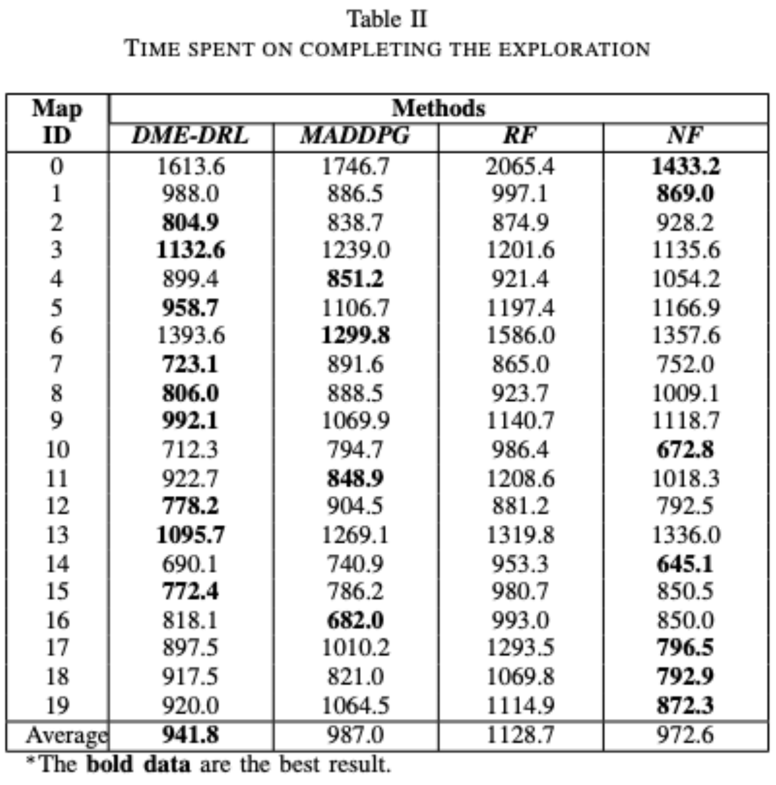


Table II
TIME SPENT ON COMPLETING THE EXPLORATION

Map ID	Methods			
	DME-DRL	***MADDPG***	***RF***	***NF***
0	1613.6	1746.7	2065.4	**1433.2**
1	988.0	886.5	997.1	**869.0**
2	**804.9**	838.7	874.9	928.2
3	**1132.6**	1239.0	1201.6	1135.6
4	899.4	**851.2**	921.4	1054.2
5	**958.7**	1106.7	1197.4	1166.9
6	1393.6	**1299.8**	1586.0	1357.6
7	**723.1**	891.6	865.0	752.0
8	**806.0**	888.5	923.7	1009.1
9	**992.1**	1069.9	1140.7	1118.7
10	712.3	794.7	986.4	**672.8**
11	922.7	**848.9**	1208.6	1018.3
12	**778.2**	904.5	881.2	792.5
13	**1095.7**	1269.1	1319.8	1336.0
14	690.1	740.9	953.3	**645.1**
15	**772.4**	786.2	980.7	850.5
16	818.1	**682.0**	993.0	850.0
17	897.5	1010.2	1293.5	**796.5**
18	917.5	821.0	1069.8	**792.9**
19	920.0	1064.5	1114.9	**872.3**
Average	**941.8**	987.0	1128.7	972.6

*The **bold data** are the best result.

Figure 19: DME-DRL published results for the number of steps

DME-DRL only published plots for distance, but Figure 20 shows it performs similarly with Nearest Frontier.

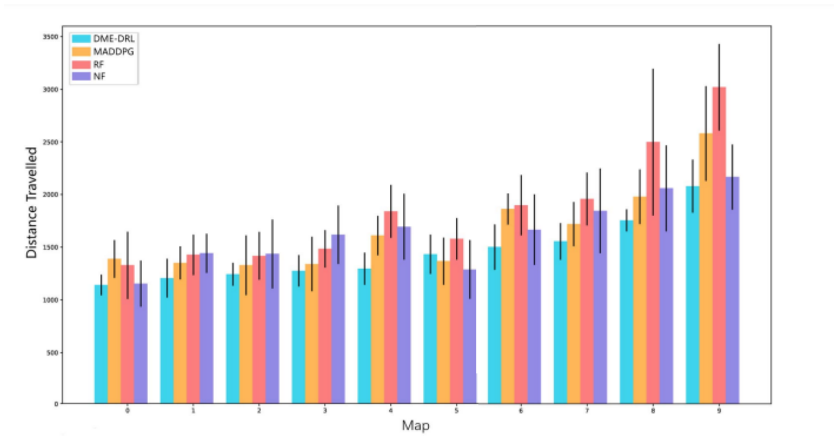


Figure 2. Comparison of Distance Travelled among DME-DRL, MADDPG, RF, and NF.

Figure 20: DME-DRL published results for distance

4.3 Local Interactions

In DME-DRL the robots come within communication range of each other as frequently as Nearest Frontier. A high number of local interactions suggests exploration may be inefficient because if robots stay close together this reduces the new area that can be explored in a time step. DME-DRL has a high number of local interactions because there is no factor in the reward function found in Equation 1 that explicitly discourages the robots from staying apart. Robots are only rewarded for exploring new area and punished for wasting time steps in exploration. Furthermore, if robots start close together, their observations will be identical at the beginning so they select the same directions, travel to the same area, and waste exploration time.

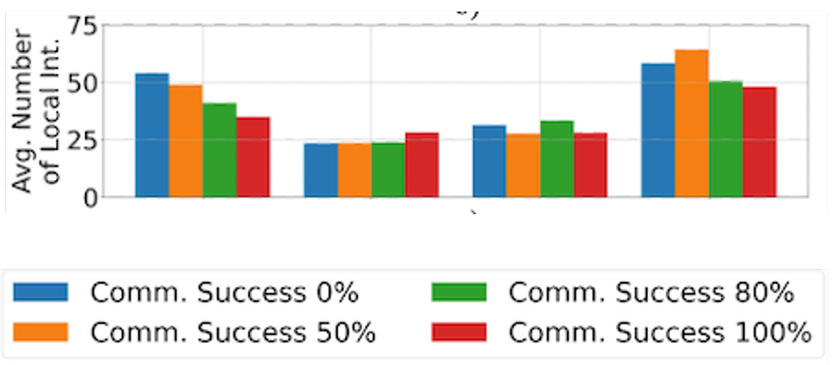


Figure 21: Average Number of Local Interactions in the Test Environment

4.4 Objective Function

We defined an objective function in Equation 4 to better measure the efficiency of exploration. On a percentage of area explored vs joint distance travelled basis, DME-DRL came second in efficiency to the Planning Based Method in Figure 22. This is promising because if we could improve the training process, DME-DRL may yield more efficient explorations.

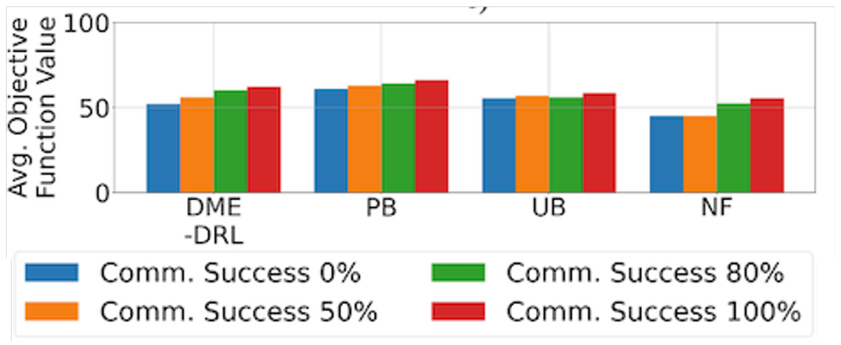


Figure 22: Average Objective Function Values in the Test Environment

It becomes more clear that DME-DRL initially lags in exploration rate from its weakness compared to its peers but finishes right after the Planning Based method in Figure 23.

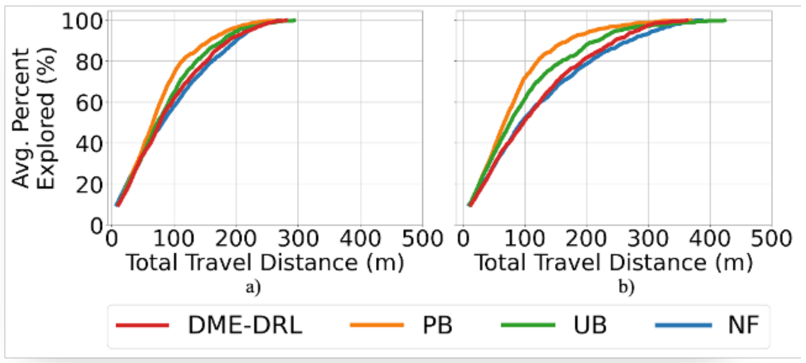


Figure 23: Interpolated Percentage of Area Explored vs Total Distance Travelled in the Test Environment

4.5 Parallel World

In this subsection we describe our attempt to program the robots to move in parallel for accurately modelling an exploration mission. We made significant changes to the codebase, but initially were unable to train the parallel algorithm. We noticed the parallel algorithm took 5000 time steps but was unable to complete an exploration episode in Figure 24. In comparison, our sequential sequential algorithm took 70 steps to complete and exploration episode during training.

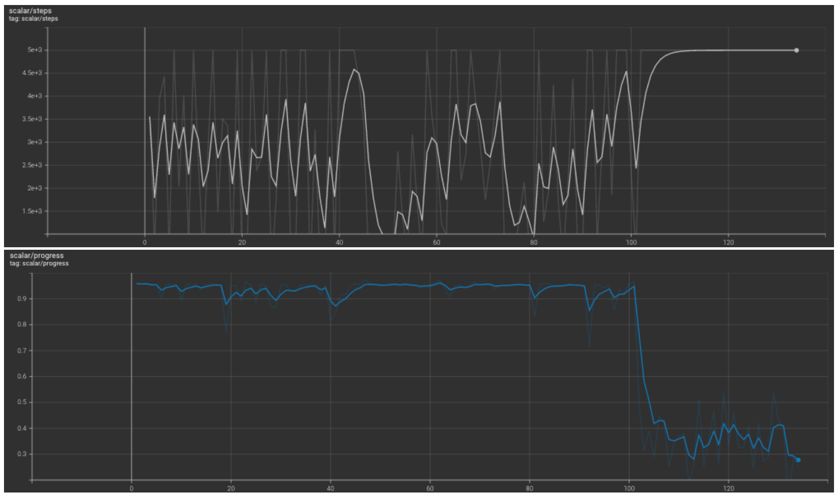


Figure 24: Number of steps (top) and percentage of area explored at the end of each episode (bottom) during training of a parallel algorithm without noise.

We found that this occurred because robots would get stuck against a wall and repeatedly move in the direction of the wall. This is because DME-DRL does not encourage exploration during training, robots always exploit the best action that they have learned. However, early in training they would be learning from a random policy. To remedy this problem, we introduced noise to the action selection so that robots could break free from this predicament by taking a different action to randomly explore its options [5]. This noise would decay over the time as the robots learned a better policy. We ran many experiments varying the learning, reward and action noise parameters. With this parallel version of DME-DRL, the robots were able to complete more exploration episodes, but they still took 900 steps to complete an episode as shown in Figure 25.

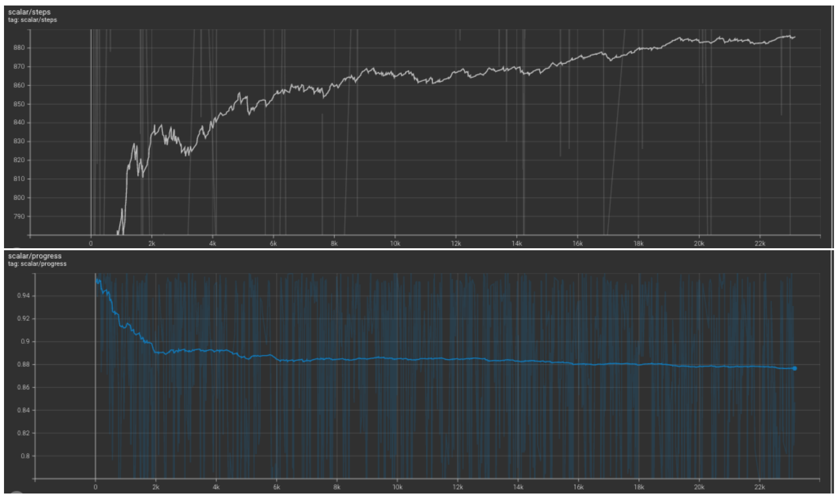


Figure 25: Number of steps (top) and percentage of area explored at the end of each episode (bottom) during training of a parallel algorithm with noise.

5 Conclusion

We achieved the two goals of our thesis: developing a simulation for USAR environments for training and testing DRL methods and benchmarking the performance of DME-DRL in this simulation. Its test results of a 6% degradation from Nearest Frontier was worse than its published performance of a 3% improvement over Nearest Frontier in the number of steps occured for three main reasons:

- DME-DRL was not designed with considerations of the probability of failed action, communication, scan
- DME-DRL was initially trained and tested in floor plans with no obstacles
- DME-DRL was tested with the robots starting close together which is known to hamper its performance

Even with this discrepancy in results, DME-DRL has fulfilled its purpose as a benchmark. We have confirmed the reproducibility problem exists in DRL, and future researchers can use our simulation and benchmark algorithms for to develop their multi-robot exploration methods.

However, DME-DRL's performance as a learning method has much to be desired in comparison to traditional methods. After all, the exploration rate results indicate DME-DRL shows promise.

For the remainder of this section, we propose several research directions for the future. It may be worthwhile to explore the fundamental question of why DME-DRL was unable to learn and improve in performance throughout training. Perhaps by further tuning the hyperparameters, adding noise to decision making during training, or revamping the network architecture design, DME-DRL could outperform traditional exploration methods.

Such an achievement may require changes to the algorithm on a fundamental level. If we could continue our work on improving the performance of our implementation of DME-DRL that programmed the robots to move in parallel. Programming the robots to move in parallel offers more opportunities for robots to communicate and possibly explore more area per time step.

In the results, we also showed DME-DRL had a high number of local interactions. This means that exploration could be inefficient because the robots were staying close together. If we redesign the reward function to punish a robot for being in the vicinity of another robot unless it can benefit from communication, we can reduce unnecessary local interactions and potentially reduce the number of time steps.

As for the local interactions which are necessary robots could share where they plan to explore during communication. This information helps the communicators coordinate and travel to different areas that would maximize the joint area.

The aforementioned communication scheme is one way we can introduce higher level planning. Instead of the robot deciding the primitive direction it should immediately head towards, higher level planning suggests that the robot should decide on where to go over a longer time horizon and better coordinate with its team members [33].

References

[1] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent & Robotic Systems*, vol. 72, 11 2013.

[2] 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.

[3] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, pp. 476–481 vol.1, 2000.

[4] C. Stachniss, O. Mozos, and W. Burgard, "Efficient exploration of unknown indoor environments using a team of mobile robots," *Annals of Mathematics and Artificial Intelligence*, vol. 52, p. 205–227, 2008.

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

[6] J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché Buc, E. Fox, and H. Larochelle, "Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program)," 2020.

[7] D. He, D. Feng, H. Jia, and H. Liu, "Decentralized exploration of a structured environment based on multi-agent deep reinforcement learning," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 172–179, 2020.

[8] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[9] R. G. Colares and L. Chaimowicz, "The next frontier: Combining information gain and distance cost for decentralized multi-robot exploration," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, (New York, NY, USA), p. 268–274, Association for Computing Machinery, 2016.

[10] A. D. Haumann, K. D. Listmann, and V. Willert, "Discoverage: A new paradigm for multi-robot exploration," in *2010 IEEE International Conference on Robotics and Automation*, pp. 929–934, 2010.

[11] M. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.

[12] R. G. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. L. S. Younes, "Coordination for multi-robot exploration and mapping," in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, p. 852–858, AAAI Press, 2000.

[13] R. Zlot, A. Stentz, M. Dias, and S. Thayer, “Multi-robot exploration controlled by a market economy,” in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 3, pp. 3016–3023 vol.3, 2002.

[14] L. Matignon, L. Jeanpierre, and A.-I. Mouaddib, “Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes,” in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI’12, p. 2017–2023, AAAI Press, 2012.

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

[16] A. Bautin, O. Simonin, and F. Charpillet, “Minpos : A novel frontier allocation algorithm for multi-robot exploration,” in *Intelligent Robotics and Applications* (C.-Y. Su, S. Rakheja, and H. Liu, eds.), (Berlin, Heidelberg), pp. 496–508, Springer Berlin Heidelberg, 2012.

[17] L. Matignon, L. Jeanpierre, and A.-i. Mouaddib, “Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes,” *AAAI 2012*, 02 2013.

[18] M. Geng, X. Zhou, B. Ding, H. Wang, and L. Zhang, *Learning to Cooperate in Decentralized Multi-robot Exploration of Dynamic Environments: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13–16, 2018, Proceedings, Part VII*, pp. 40–51. 01 2018.

[19] M. Geng, K. Xu, X. Zhou, B. Ding, H. Wang, and L. Zhang, "Learning to cooperate via an attention-based communication neural network in decentralized multi-robot exploration," *Entropy*, vol. 21, no. 3, 2019.

[20] Z. Chen, B. Subagdja, and A.-H. Tan, "End-to-end deep reinforcement learning for multi-agent collaborative exploration," in *2019 IEEE International Conference on Agents (ICA)*, pp. 99–102, 2019.

[21] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2020.

[22] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," 2016.

[23] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[25] R. C. Staudemeyer and E. R. Morris, "Understanding lstm – a tutorial into long short-term memory recurrent neural networks," 2019.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra,

S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[27] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems* (S. Solla, T. Leen, and K. Müller, eds.), vol. 12, MIT Press, 2000.

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[29] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[30] L. Tingguang, H. Danny, L. Chenming, Z. Delong, W. Chaoqun, and M. Q.-H. Meng, "Houseexpo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots," *arXiv preprint arXiv:1903.09845*, 2019.

[31] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[32] "hedingjie / DME-DRL kernel description." https://github.com/hedingjie/DME-DRL/tree/master/src.

[33] C. Amato, G. Konidaris, L. P. Kaelbling, and J. P. How, "Modeling and planning with macro-actions in decentralized pomdps," *J. Artif. Int. Res.*, vol. 64, p. 817–859, jan 2019.