Multi-Agent Search based on distributed Deep Reinforcement Learning

Weiheng DAI*
e0452827@u.nus.edu
Mechanical Engineering
National University of Singapore

Guillaume SARTORETTI
guillaume.sartoretti@nus.edu.sg
Mechanical Engineering
National University of Singapore

ABSTRACT

This project considers the problem of aerial swarms search with a group of agents with limited sensing and communication capabilities distributed over a given domain. This domain is discretized into unit cells, each associated to a probability and uncertainty of target presence, thus constructing an *information map*. Each agent keeps a local belief of the this information map, which they can update by moving and sensing their surroundings. We further propose a conventional communication method that allows nearby agents to merge their local maps to obtain relevant information about areas they have not yet visited. Specifically, we treat this collaborative task as a decentralized multi-agent path finding (MAPF) problem and train the agents to make individual decisions via a policy represented by a deep neural network. The goal of the agents is to find all the hidden static targets as soon as possible. In this article, we demonstrate the resulting decentralized search process in simulation. Moreover, by comparing the result for search with communication and without communication, we highlight how collaboration can help improve the agents' performance.

KEYWORDS

Multi-agent systems, Deep reinforcement learning, Multi-robot search, Decentralized control, umerical simulations

1 INTRODUCTION

The rapid growth of digital information, high-speed computing and the development of affordable unmanned aerial vehicles (UAVs) with embedded visual sensing make it possible to carry out some collaborative tasks that can involve hundreds of autonomous robots. Large-scale multi-robot system can help automate numerous key civil and military applications such as area surveillance, environment reconstruction and dangerous area surveys. One such application is aerial swarms search for unknown targets, for which many approaches have been proposed to date. The most straightforward way to approach this problem is through geometric, which is very useful when lacking any a priori information about the likely positions of targets [1][2]. On the other hand, when such a priori information is available, decentralized or gradient-based method can be applied by exploiting the probability distribution that describes the likely target positions [3][4][5][6]. However, information gradient is sensitive to noise and can drive agents to local maxima, which also reduces efficiency. When we want to make full use of the information and find a more optimal solution in a long-term way, optimization-based methods can be used [7][8]. Through maximizing the gathered information, agents can work in a more efficient way but since these methods are usually centralized, they lose scalability to larger teams. In this project, we seek a decentralized solution to multi-agent collaborative search where agents have individual beliefs about the world and make individual decisions.

More specifically, we break the aerial swarms search problem into three parts. We first formulate the problem from a real environment into a discretized model. We divide the world into a discrete two-dimensional map making up of a number of unit cells, and each cell contains some information comprised of probability, uncertainty, agent position, and target status. The probability relates to the likelihood of target existence. The uncertainty represents how confident we are about the information levels in a given area. As for the agents, we give each the ability to do several specific actions to move to an adjacent cell depending on whether they are allowed to move diagonally. Agents also have some visual and sensing characteristics, and they change the environment and update their local information map accordingly.

With the problem formulated, we then extend the distributed RL framework on Pathfinding via Reinforcement and Imitation Multi-Agent Learning (PRIMAL) from Sartoretti et al. [9] and cast the search problem as a path planning problem. In PRIMAL, agents learn a decentralized policy to plan a path and arrive at their goal positions. Similarly, in our search problem, goals are implicitly defined as areas with high probability, and agents learn to execute actions to get them close to goals and search around. Through constructing a deep neural network that maps agents' local observation to their next action, we propose a new decentralized, scalable solution to multi-agent search.

We finally show how to improve the agents' search time and accuracy by relying on explicit communications among agents. Specifically, we propose to let agents share information that will allow each other to update their local maps to acquire information related to areas they have not visited. By rewarding agents for such helpful communications, agents can learn to cooperate and make more informed decisions.

The article is structured as follows: In section 2, we introduce the search related work and MAPF based on deep reinforcement learning (RL). We formulate the search problem and cast it into a RL problem in section 3. How we randomize the environment and build the network are described in section 4. Our communication method for search is illustrated in section 5. In section 6 and 7, we list the result of the experiment and present concluding remarks.

2 PRIOR WORK

This section focuses on previous works on multi-robot search methods, as well as MAPF based on deep reinforcement learning (DRL).

In this work, we are inspired by existing search methods, and present a new method that relies on DRL to improve search efficiency and scalability.

2.1 Search

Multi-agent search is a central robotics problem, which considers search for different kinds of targets and under different conditions. Many search strategies have been summarized in [10], which illustrate the underlying theory in single-agent searching for single or multiple, static or moving targets. Furthermore, collaborative search has been drawing researchers' interests [11][12] [13][14][15], whose methods have been relying on a variety of tools such as team-optimal, dynamic programming and distributed control.

In general, there are two broad classes of search methods, depending on the availability of a priori information about the likely location of targets. Methods like geometric coverage are applicable in some situation where no information can be acquired, and agents move to cover all the area of this region.

However, more advanced collaborative search methods can exploit a prior information if it available. First, gradient-based method have been proposed in [1][2][16], from the likelihood of targets, agents search greedily by driving agents to local information maxima. Masoud et al. proposed a PSO-based multi-robot cooperation method to search targets where they assume target emits a signal like heat for the robot to sense it and determine a locally favorable direction. PSO does not use the gradient of the problem being optimized, and it does not guarantee an optimal solution. There are also many other decentralized search strategies. Chung et al. formulated a multi-agent decision-theoretic of probabilistic search problem [17]. Furukawa et al. [18] presented a control technique which use recursive Bayesian filtering to autonomously search and track multiple targets with distributions and probabilistic motion models. These methods can not be applied to real-life scenarios easily with large agent teams, and our work looks for scalable, decentralized multi-agent search methods to control the agents searching more intelligently.

2.2 Multi-agent Path Finding (MAPF)

MAPF is an NP-hard problem even when approximating optimal solutions. Here we want to emphasis the decentralized learning MAPF planner where agents learn their own policy and can be implemented to large-scale multi-agent system easily. Some DRL-based MAPF decentralized planners show great potential in solving MAPF problem, for example, Sartoretti et al. [9] proposed pathfinding via reinforcement and imitation multi-agent learning (PRIMAL), a new framework for decentralized, reactive MAPF. They showed that PRIMAL worked well in low obstacle densities situations, which combined the advantages of distributed reinforcement learning and imitation learning. As we want to solve the multi-agent search problem in a decentralized way, such distributed learning based approaches seem like a good starting point for us to base our work upon.

3 BASIC DEFINITIONS AND ASSUMPTIONS

Given a real environment with several static targets and some information about the search domain, we want to use a team of autonomous mobile agents (e.g., UAVs) to find as many targets as possible within a limited time. We formulate this task to a mathematical problem and cast this problem to a reinforcement learning (RL) problem.

3.1 Problem Formulation

We consider a discrete search domain divided into uniform cells. The size of the domain determines the number of cells, each representing a fixed area. Without restricting the generality of our approach, we consider a square region composed of $n \times n$ cells to represent the search domain. The position of each cell in the region can be represented as $[x, y]^T$. The probability of target existence in each cell is $p(x, y) \in [0, 1]$ defining the 2D *probability map*, and uncertainty for each cell is $u(x, y) \in [0, 1]$ similarly defining the *uncertainty map*. These two maps are referred to as the *information map*.

The agents are the only movable object in this region. All of them are assumed to move in a fixed plane whose size is the same as the region. In this plane, they must avoid collisions with each other or obstacles. The positions of the N agents are denoted as $a_i = [x_i, y_i]^T$, with $i = (1, ..., N)$.

Additionally, agents have a *sensor footprint* which is assumed to be a circle region Γ. Agents can update the information in this region and stochastically detect targets using this footprint. Γ is defined as a 5×5 matrix (where only a circular footprint is non-zero) as

$$\Gamma = \begin{bmatrix} 0 & 0 & 0.1353 & 0 & 0 \\ 0 & 0.3679 & 0.6065 & 0.3679 & 0 \\ 0.1353 & 0.6065 & 1.0000 & 0.6065 & 0.1353 \\ 0 & 0.3679 & 0.6065 & 0.3679 & 0 \\ 0 & 0 & 0.13530 & 0 & 0 \end{bmatrix}.$$

Values in Γ represent the probability of detecting a nearby target relative to the agent's position ($\Gamma_{3,3}$) at each tiem step. For example, if a target is beneath the agent, then the agent will always find it ($\Gamma_{3,3} = 1$), but for slightly further targets, the probability of detection decreases.

Additionally, agents can update the levels of probability and uncertainty in their surroundings according to $\Gamma_{x,y}$. The collected information in each cell of their sensor footprint are $\Delta p(x, y)$ and $\Delta u(x, y)$ as equation 1 and 2. With the agent collecting the information, the information in this environment will also decrease by the same amount. That is, after an agent passing by, if the probability in a cell is collected by the agent, then it becomes to $p'(x, y) = p(x, y) - \Delta p(x, y)$, and the same for uncertainty.

$$\Delta p(x, y) = \lambda k \Gamma_{x,y} p(x, y), \tag{1}$$

$$\Delta u(x, y) = \lambda k \Gamma_{x,y} u(x, y)), \tag{2}$$

where k is a coefficient related to world size n, λ is a constant coefficient to control the amount of information removed per time step (experimentally chosen to be 0.001). That is to say, agents can detect targets and decrease information contents in each cell of the region in accordance with their sensor footprint.

3.2 Search as a RL problem

In this section, we cast multi-agent search as a reinforcement learning problem. To this end, we describe the agent's observation space and action space how to calculate the reward for each action.

3.2.1 Observation Space. Agents start their search with a local copy M_i of the global priori information map, where M_i contains the probability map and uncertainty map.

In real-life situations, UAVs cannot observe the whole search domain by relying on onboard sensing only. What is more, some area in their field of fiew (FOV) may be blurred. Therefore, we believe such partial observability of the world with an explicit area is realistic. Moreover, the limited FOV can reduce input dimensions in the training process in a neural network, which can help to save training time and improve generalization to domains with arbitrary sizes.

Considering the visual system of UAVs, we define their observation FOV as a square area Θ (in practice, 11×11) centered around themselves. At every time step, agents can only extract the portion of their local *probability map*, *uncertainty map* corresponding to Θ FOV. In this FOV, agents can see each other and surrounding obstacles. Additionally, we define an explicit area Φ centered around agents and outer area represents the blurred area in their FOV. Agents can directly sense the environment around them within Φ and update the probability and uncertainty in their local maps before they extract the observation. In this project, we investigate the effect of various sizes of Φ (11×11 or 5×5) and compare the results.

The extracted information is reconstructed into one matrix with five channels below, and each channel contains a 2D 11×11 matrix.

- other agents nearby are represented as 1, and empty cells are considered as 0
- obstacles in their FOV which also contains the area out of boundary
- location of targets which have been found by any agent
- $p(x, y)$ extracted from their local map, probability levels out of boundary is assumed 0
- $u(x, y)$ extracted from their local map, uncertainty levels out of boundary is assumed 0

We also add a 3×1 vector of real value containing agent's position and the number of remaining targets to help agents know what is the status of the search task. All these channels and the scalar construct agent's observation which help agents determine their next action.

3.2.2 Action Space. Agents have two action modes containing moving in the four cardinal directions or adding extra four intermediate directions. In mode one, agents can take at most five different actions based on their current position. These actions correspond to moving in the four cardinal directions (north, east, south, west) and staying still. In mode two, they can additionally move to four intermediate directions so totally nine actions. At each time step, all the agents execute a single action that is individually decided. The probability map and uncertainty map of the environment are then updated based on their sensor footprint Γ.

3.2.3 Rewards Structure. The reward structure used in this work is similar to that in many gridworlds where agents are penalized for each time step, in order to encourage them to finish a given task as soon as possible.

The probability and uncertainty have a close relationship to the target position, which means collecting more information is correlated to finding more targets. We introduce a reward from collecting information to encourage agents to explore the world and help them learn how to locate and find targets. We also introduce a collision penalty to train agents to move around each other safely. For finding a targets, we give agents a relatively high reward. The sum value of all these rewards listed in table 1 will be the total reward at each time step.

The Collect Information reward R_{in} is calculated as below to keep R_{in} relating to the amount of information an agent collects within their sensor footprint at each time step:

$$M_p = \sum_{x=1}^{5} \sum_{y=1}^{5} \Delta p(x, y), \tag{3}$$

$$M_u = \sum_{x=1}^{5} \sum_{y=1}^{5} \Delta u(x, y), \tag{4}$$

$$R_{in} = \frac{1}{\lambda} \xi (M_u + M_p), \tag{5}$$

where the ξ is a constant coefficient. By normalizing the reward R_{in}, the final cumulative reward for one episode can be kept approximately to a constant, thus helping stabilize the training process and allow comparisons between different training runs.

Table 1: The Reward Structure

Actions	Reward
Move(N/E/S/W/None)	-0.05
Agent Collision	-0.2
Find Target	+10
Collect Information	R_{in}

4 POLICY REPRESENTATION

With all the assumptions defined in section 3, this section details how we randomize the environment maps as well as construct the neural network used to represent the agents' policy.

4.1 Environment

As one of the essential structures where agents learn their collaborative policy, we here show how these information of the environment are randomized during training.

4.1.1 Episode. When we train the agents, we will create many episodes until the policy converges. If agents find all targets or time step counts to 256, we define it as the termination of an episode. At the beginning of each episode, we create a new random environment.

4.1.2 World map. The world size is set to be a random number within [20, 128] to vary the difficulty of search task. For larger world, agents should cover more areas to find the targets. Agents need to learn how to work collaboratively in different world size. The number of agents, on the other hand, is not changed and remains N=8 during training.

4.1.3 Probability Map and Uncertainty Map. As the only information which is related to target location available for agents, we randomize these maps at the beginning of each episode.

We want to use a multi-model map with several local maximas to represent the information, so we create a map containing several Gaussian distributions. Considering the world size n in the previous section, we add m ($m \in [16, 32]$) Gaussian distributions into the map. For each distribution, the value in each cell of the discrete map can be calculated as equation 7 below:

$$f_m(x,y) = \frac{e^{-\frac{1}{2(1-\rho_m^2)}\left[\frac{(x-\mu_{Xm})^2}{\sigma_{Xm}^2}+\frac{(y-\mu_{Ym})^2}{\sigma_{Ym}^2}-\frac{2\rho(x-\mu_{Xm})(y-\mu_{Ym})}{\sigma_{Xm}\sigma_{Ym}}\right]}}{2\pi\sigma_{Xm}\sigma_{Ym}\sqrt{1-\rho_m^2}}, \tag{6}$$

where μ_{Xm} and μ_{Ym} are the mean of the distribution, which also represents the position of the maximum in this distribution; ρ_m is correlation between X and Y; σ_{Xm}^2 and σ_{Ym}^2 express the area of the distribution. We set the variance for the probability map to a random number $(0, 10n)$ related to the world size n.

The resulting probability map might still contain areas with near-zero levels, which we would like to avoid to encourage exploration of the whole domain, thus we add a baseline with η percent of the sum probability and evenly allocated it to every cell in order to encourage the agent to explore more in our training process.

Furthermore, we normalize this probability map using equation 7, so all the probability sums up to 1, and each cell is represented as $p(x, y)$ in equation 7 as:

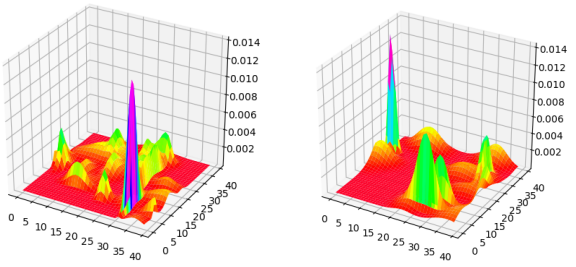
$$p(x,y) = \frac{1}{\sum_{i=1}^{m} f_i}\left((1-\eta)\sum_{i=1}^{m} f_i(x,y) + \frac{\eta\sum_{i=1}^{m} f_i}{n^2}\right), \tag{7}$$

$$u(x,y) = \frac{1}{\sum_{i=1}^{m} f_i}\sum_{i=1}^{m} f_i(x,y), \tag{8}$$

where $f_m = \sum_{x=1}^{n}\sum_{y=1}^{n} f_m(x,y)$. The uncertainty map is created in the same way as the probability map except that the variance of uncertainty map is $(0, 20n)$ and does not have a baseline uncertainty level. The $u(x, y)$ is formulated in equation 8.

The probability and the uncertainty maps are shown in Fig.1(a) and 1(b).

4.1.4 Targets. We distribute a random number j of static targets on the map (in practice, $j \in [8, 16]$). Our goal there is to randomly place targets by relying on both the probability and uncertainty maps. We rely on a two-stage system: first, we randomly sample a tentative target position (x, y) according to the probability distribution $p(x, y)$ over the domain. Second, we perform a uniform random draw, and place the target at (x, y) if that draw is smaller than $1 - u(x, y)$ (i.e., larger local uncertainty decreases the probability a target is placed). Otherwise, we sample a new tentative target position (x', y'). We repeat the process until all targets are placed. This process ensure the location of the targets is coherent with the



(a) Probability Map (b) Uncertainty Map

Figure 1: Two kinds of information map. The blue/purple maximum area represents the high probability or uncertainty

a prior information in the agents' information maps, allowing them to learn to perform informed individual decisions.

4.2 Network Structure

We use a deep neural network to approximate the agent policy, which maps an agent's observation to the agent's next action. This network takes inspiration from VGGnet [19] and extends our prior works on distributed learning for MAPF [20]. Our network is mainly made up of a 6-layer convolutional network, two max-pooling layers with several 3×3 kernels between each, five full connected layers, and a long-short term memory (LSTM) cell. The structure is in Figure 3. To update the neural network, we extend asynchronous advantage actor-critic (A3C) algorithm [21] shown in Figure 2, where agents interact with environment individually and then update the global network.

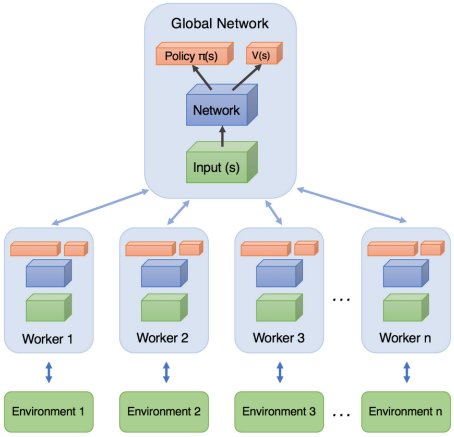


Figure 2: Diagram of the A3C architecture [22]

The inputs to the neural network are agents' observed information contain two separate parts, matrical and scalar information, as mentioned in section 3. These two parts of inputs are processed independently and simultaneously, then concatenated together for processing in the last layers of the network. The concatenation

of processed inputs is passed through two full-connected layers and an LSTM cell. The output layers consist of five (or nine) policy neurons with softmax activation which produces the probability of all the actions.

In the training process, the policy, value outputs are updated in batch mode when an episode finishes. What we want is to minimize the loss L_v through updating the value to match the total discounted reward R_t as:

$$R_t = \sum_{i=0}^{k} \gamma^i r_{t+i}, \tag{9}$$

$$L_V = \sum_{t=0}^{T} \left(V\left(o_t;\theta\right) - R_t\right)^2. \tag{10}$$

We use advantage estimates rather than just discounted returns, which not only allows the agent to determine how good its actions were but also how much better they turned out to be than expected. The advantage function is defined as $A\left(o_t, a_t;\theta\right)$. The policy loss L_π is used to obtain gradients and update the neural network. We add an entropy term $H(\pi(o))$ to policy loss, which helps to decrease premature convergence. The equation is as follows:

$$A\left(o_t, a_t;\theta\right) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V\left(o_{k+t};\theta\right) - V\left(o_t;\theta\right), \tag{11}$$

$$L_\pi = \sigma_H \cdot H(\pi(o)) - \sum_{t=0}^{T} \log\left(P\left(a_t|\pi, o;\theta\right) A\left(o_t, a_t;\theta\right)\right). \tag{12}$$

5 COMMUNICATION

To improve performance when the agents carry out collaborative task, we propose a communicate mechanism to help the agents acquire information about areas they have never visited.

5.1 Communication Principle

We define a communication area whose size is the same as the sensor footprint Γ (5x5 around the agent's position). Agents are only able to communicate with other agents inside this communication area. Each agent has a local information map M_i. During communication, agents exchange their local map with each other. The minimum probability and uncertainty in each cell of agents' local map M_i can represent the most recent information they have about the environment, because at each time step, agents can collect information from the environment which decreases these levels in their local maps. The smaller the value in their local map M_i, the closer this value is to the "true value". Therefore, for each map in M_i we formulate the communication process as equation 13:

$$M_i[p(x,y)] = min(M_i[p(x,y)], M_k[p(x,y)], ...M_{k-n}[p(x,y)]), \tag{13}$$

$$M_i[u(x,y)] = min(M_i[u(x,y)], M_k[u(x,y)], ...M_{k-n}[u(x,y)]), \tag{14}$$

where $k, k-1, ..., k-n$ represents nearby agents' ID, and the information Through this approach, agents can acquire the latest information from the group of neighbor agents and update their local maps. Example changes in an agent's local information map during communication is shown in Figure4(a) and Figure4(b).

5.2 Reward Modification

Every time an agent communicates with others, total probability and uncertainty in their local maps will be updated by ΔM_p and ΔM_u with:

$$\Delta M_p = \sum_{x=1}^{n}\sum_{y=1}^{n} \left(M_{i0}[p(x,y)] - M_{i1}[p(x,y)]\right), \tag{15}$$

$$\Delta M_u = \sum_{x=1}^{n}\sum_{y=1}^{n} \left(M_{i0}[u(x,y)] - M_{i1}[u(x,y)]\right), \tag{16}$$

where M_{i0}, M_{i1} represent the agent's local map M_i before communication and after communication. To encourage the agents to carry out more useful communication and get more ΔM, we add a reward R_{comm} to reward structure Table 1. This communication reward R_{comm} can be computed at each time by equation below.

$$R_{comm} = \frac{1}{\lambda}\tau(\Delta M_p + \Delta M_u) \tag{17}$$

where the τ is a coefficient.

6 RESULTS

In this section, we illustrate training details and results of our DRL search controllers. We test four different models of our controllers with different numbers of agents under the same set of random environments. Additionally, we compare the search results of our method with other controllers like information surfing in [5][6] to demonstrate the superiority of our method.

6.1 Training Details

For our DRL controllers, we have trained four different models under the assumption and definition we have in section 3 and 4, containing the large explicit area model, small explicit area communication model, small explicit area no communication model and diagonal action model. In each model, we only change the explicit area or whether communicating or whether they can move diagonally, which makes the results easily to compare.

6.1.1 Performance Metric. During the training process, we record some key data to represent the performance of different models. We define them as follows.

- Average search time L: It is average time the agents use to find all the targets. It has an upper limit at $L = 256$, which means agents fail to finish at completing the search task.
- Reward R: The reward value is accumulative in every episode, which sums agents' action rewards at every step throughout each episode.
- Communicating frequency F: Every time two agents exchange their local information maps, the communication frequency increases once. This value represents how many times the agents communicate with others in one episode.
- Information collection per step $\bar{M}$: Agents will collect some amount of information shown in equation 3 and 4 at each time step, and $\bar{M}$ is the mean of M_p or M_u for each time step.

6.1.2 Different DRL Models. Next, we introduce the details of our DRL controller. We train a benchmark model *model A* with the largest explicit area $\Phi = 11 \times 11$, and no communication between

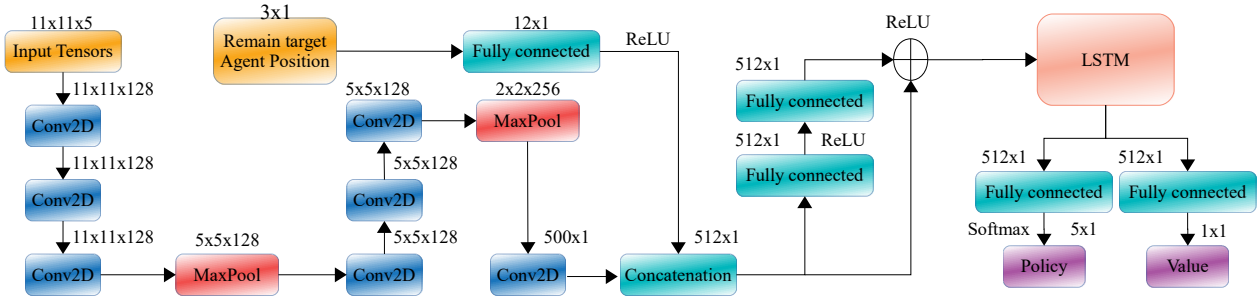
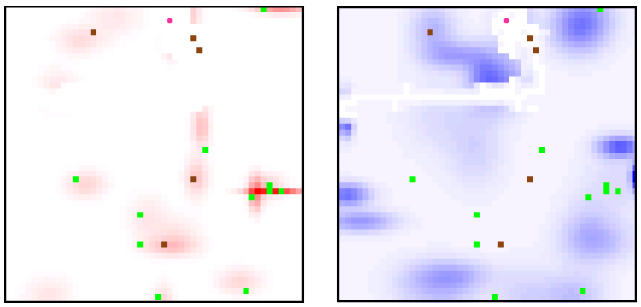
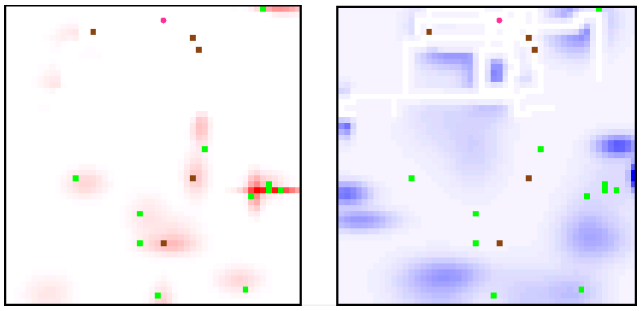


Figure 3: The neural network structure



(a) Before communication



(b) After communication

Figure 4: Visualization of communication. Red areas represent high probability, and blue areas represent high uncertainty. We draw each agent as a purple round. Targets having been found are the brown square, and the remaining targets are the green square. Local information map of the agent near the upper boundary is (a), after communication, the color in many areas of the map changes to white, which means no information is here anymore.

agents. The other parameters are the same as those described in section 4. The size of the agent's explicit area is the same as its FOV, which means information in each cell of agent's observation is equivalent to the true levels. This model is expected to perform best in general.

In order to measure the performance improvement from communication, we reduce the agent's explicit area Φ from 11×11 to 5×5. This model name is *model B*. In this model, information within the agent's explicit area is still up to date, but the outer area of its observation may be outdated. Although agents cannot obtain all the latest information directly from the environment, they can use communications to obtain more reliable information about outdated areas from other agents. In this way, we hope that agents can learn to cooperate and exchange their local maps to get more reliable information.

In addition, we define a *model C*, whose parameters are the same as those in *model B*, except that no communication happens. In order to know the levels of communication compared to *model B*, we still count the communicating frequency F to record the total times that agents meet others within the communication area (sensor footprint). This model is expected to be the worst.

Furthermore, we add a diagonal movement model where agents can move diagonally to increase the scalability in real scenarios. We also add the communication mechanism. Compared to *model B*, we only change it to action mode two to allow agents to move diagonally.

For each model, we trained about 9000 episodes until the reward R converged. Through the reward curve in Figure 5, we can find the policy nearly trained after about 6000 episodes.

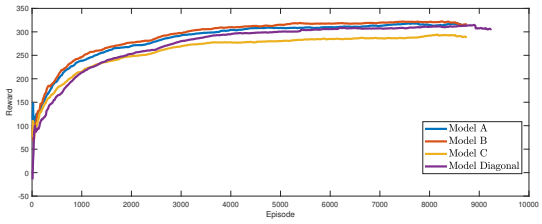


Figure 5: Reward curve for each configuration

6.1.3 Information Surfing. We also mix several search methods based on decentralized Bayesian and gradient-based as the information surfing method in order to compare with our DRL method. We use several vector containing the gradient direction and agent force to produce the agent nex action. The gradient help to agents to find the local minima, and the agent force base upon Coulomb's law of electrostatic force to avoid agent collision.

6.2 Results of Training

In this section, we analyze the key data from the models mentioned in the previous section. The average search time L, information collection per step $\bar{M}$ and communicating frequency F are relatively shown in Figure 6.

The average search time directly reflects the success rate during training process. The smaller the L, the higher the success rate of the policy (i.e., all targets were found quickly). For these four models, the *model diagonal* has the shortest average search time compared with other models, because agents can learn to move a longer distance at each time step. The second shortest is *model A*, where agents have more accurate information in their observations. Communication *model B* is also superior to no communication *model C*, which confirms that agents benefit from the communication.

Comparing the information collection $\bar{M}$ and average search time L in Figure 6, we can easily notice that when agents collect more information at each time step, they can finish the episode sooner, which means that agents finding a target is closely related to information they collect, as expected.

From two models with communication and two models without communication, we find that the communicating frequency has increased significantly in *model B* and *model diagonal*. In both models, agents prefer to communicate with other nearby agents, which also helps them to collect more information and find targets faster, although only marginally.

We can draw the following conclusion: Agents can make a good use of the information to find targets, and if the input of the neural network (the agent's observation) contains more accurate information, the agent can make better decisions through policy. What is more, communication can help agents to obtain information if they only have a small explicit area.

6.3 Result of Testing

In order to validate the performance of our model in different environments, we randomly create 3,000 maps containing different numbers of agents and targets and information maps. The details are shown in Table 2. For each map, we test the policy for several times and record the percentage of found targets to the total number of targets.

In our test, we want to avoid unrealistic cases involving too high agent densities over the domain, thus we set some restrictions, for example, in a map with world size $n = 20$, the number of agents N can only be lower than 32, similarly, for $n = 40$ we have $N \leqslant 64$, for $n = 80$ $N \leqslant 128$, and for $n = 160$ $N \leqslant 256$. The output of the policy is a matrix containing the activation level for each of the five/nine actions. We repeat the same search scenario 5 times. What's more, for these 3000 maps, we also implement information surfing (and geometric coverage) to compare with our DRL method.

Table 2: The Test Configuration

World Size n	20	40	80	160
Number of Targets N	4	8	16	32
Range of Distributions m	8	16	32	64

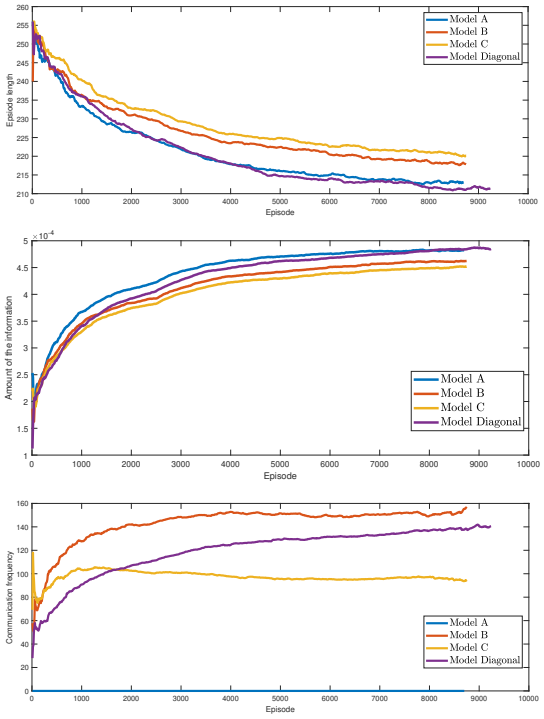


Figure 6: The average search time L, information collection per step $\bar{M}$ and communicating frequency F of the four models.

For each map set, the average percentage of found targets using four DRL search controller models are listed in Table 3. We particularly show the result when $n = 160$ in Figure 7. The comparison of our DRL method and information surfing is shown in Figure 8.

Through the testing result of the four models, we can make several conclusions. Firstly, our DRL search controller realizes more than 95% target found percentage in a limited time if the number of agent is sufficient.

Secondly, the policy using communication (*model B* and *model diagonal*) are better than no communication policy (*model C*). Communication can help agents obtain more reliable information from the observation, but only marginally. From results of testing, agents can find 3% more targets than *model B*.

Additionally, noise can also help to improve the performance. When the number of agents is relatively large ($N > 32$), the communication model even has a better performance than large explicit area *model A*. We believe this can be explained by a simple example, an agent is searching around an high information area in environment. If they miss a target, the information around the target will still be decreased by the agent's passage. When another agent comes here, this second agent will observe a potentially lower information level, and might not search the area as thoroughly, thus making the target increasingly difficult to find, so this missing target is hard to be found. On the other hand, in the communication model, outdated information can sometimes act as noise on top of the agents' observations, stimulating other agents continue searching

Table 3: Testing result by percentage of found targets

Model A	n=20	n=40	n=80	n=160
N=1	0.5190	0.3248	0.1436	0.0608
N=2	0.6610	0.5118	0.3040	0.1189
N=4	0.7815	0.7815	0.5075	0.2209
N=8	0.9220	0.9298	0.7159	0.3863
N=16	0.9875	0.9838	0.8884	0.6005
N=32	0.9940	0.9900	0.9484	0.7851
N=64	NaN	0.9923	0.9723	0.8976
N=128	NaN	NaN	0.9858	0.9443
N=256	NaN	NaN	NaN	0.9608
Model B	n=20	n=40	n=80	n=160
N=1	0.5110	0.2923	0.1493	0.0616
N=2	0.6265	0.5050	0.2908	0.1153
N=4	0.7535	0.7678	0.4639	0.2014
N=8	0.9165	0.9053	0.6938	0.3664
N=16	0.9890	0.9775	0.8703	0.5729
N=32	0.9945	0.9890	0.9580	0.7871
N=64	NaN	0.9928	0.9893	0.9219
N=128	NaN	NaN	0.9927	0.9720
N=256	NaN	NaN	NaN	0.9878
Model C	n=20	n=40	n=80	n=160
N=1	0.5320	0.2878	0.1551	0.0531
N=2	0.6690	0.4655	0.2703	0.0975
N=4	0.7660	0.7430	0.4478	0.1806
N=8	0.9240	0.8790	0.6551	0.3383
N=16	0.9700	0.9538	0.8323	0.5336
N=32	0.9615	0.9793	0.9241	0.7484
N=64	NaN	0.9783	0.9601	0.8831
N=128	NaN	NaN	0.9679	0.9356
N=256	NaN	NaN	NaN	0.9520
Model Diagonal	n=20	n=40	n=80	n=160
N=1	0.5995	0.3415	0.1859	0.0711
N=2	0.7025	0.5815	0.3175	0.1400
N=4	0.8130	0.8230	0.5446	0.2532
N=8	0.9370	0.9445	0.7589	0.4576
N=16	0.9860	0.9728	0.8930	0.6598
N=32	0.9915	0.9823	0.9473	0.8364
N=64	NaN	0.9875	0.9745	0.9353
N=128	NaN	NaN	0.9858	0.9618
N=256	NaN	NaN	NaN	0.9778

around. This noise helps to decrease the number of missed targets and improve the performance.

When we compare our DRL method with information surfing, our DRL search controller performs much better. In information surfing, agents just greedily move towards the direction where the information seems to increase, but they are easily stuck in high information area until they collect all information. This method also drives agents to local maxima, which is hard to improve.

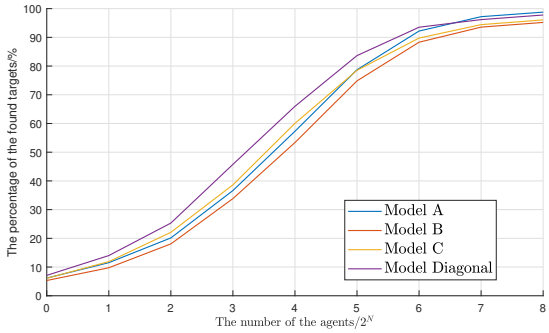


Figure 7: Average percentage of found targets for different quantity of agents in a 160×160 map

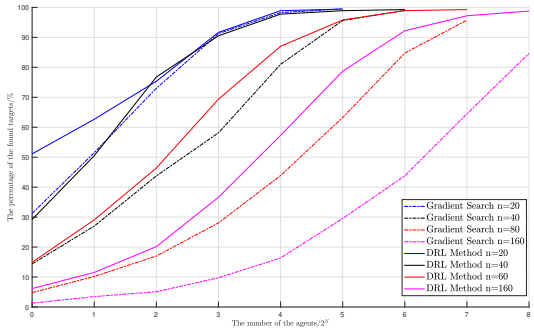


Figure 8: Comparison of average percentage of found targets between information surfing and DRL method

7 CONCLUSION

In this article, we propose an approach combining reinforcement learning and the probability models to drive agents to make decisions independently and search efficiently over a domain, which can also be implemented in real-life scenarios easily. Through comparing the results, we conclude the communication leads to a better performance in this collaborative search task.

Our approach also have some drawbacks, in our problem formulation, the (randomized) positions of the targets depends to a large extent on the information map. Therefore, the agent's behavior can also be regarded as searching high probability area in their FOV. In real-life situations, if we do not have information or only incorrect information about the search domain, our approach will likely behave poorly.

Secondly, agents have an individual map in their own memory, of which they can only extract information within their FOV at each time step. Therefore, our approach does not allow agents to fully utilize their local information map, e.g., to make longer-term decisions about their paths in the domain. This is also a disadvantage for our search controller.

To improve this disadvantages, we started to conceptualize an approach to use an attention mechanism [23][24][25]. With this mechanism, agents can choose where to place their limited FOV

over their whole information map. Then they can get more information which is no longer limited in the area around them and make decisions more pertinently.

ACKNOWLEDGMENTS

Without the patient guidance and previous work form my supervisor Prof. Sartoretti, this project would not exist. I am also grateful to Prof.LENG who is my examiner for this project. Although the period of my project was filled with many ups and downs, my effort in the multi agents search was worth it. I'm eternally grateful to the Multi-Agent Foraging group who share their source code with me, which speeds up the project progress. I would also like to thank all the members of MARMot LAB. We are the most hard-working groups in the school. Finally, the experience to learn at National University of Singapore is definitely unforgettable.

REFERENCES

[1] Howie Choset. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1-4):113–126, 2001.

[2] Vitaly Ablavsky and Magnús Snorrason. Optimal search for a moving target-a geometric approach. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 4060, 2000.

[3] Pablo Lanillos, Seng Keat Gan, Eva Besada-Portas, Gonzalo Pajares, and Salah Sukkarieh. Multi-uav target search using decentralized gradient-based negotiation with expected observation. *Information Sciences*, 282:92–110, 2014.

[4] El-Mane Wong, Frédéric Bourgault, and Tomonari Furukawa. Multi-vehicle bayesian search for multiple lost targets. In *Proceedings of the 2005 ieee international conference on robotics and automation*, pages 3169–3174. IEEE, 2005.

[5] Joseph L Baxter, EK Burke, Jonathan M Garibaldi, and Mark Norman. Multi-robot search and rescue: A potential field based approach. In *Autonomous robots and agents*, pages 9–16. Springer, 2007.

[6] F. Bourgault, T. Furukawa, and H. F. Durrant-Whyte. Coordinated decentralized search for a lost target in a bayesian world. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 48–53 vol.1, 2003.

[7] Elif Ayvali, Alexander Ansari, Long Wang, Nabil Simaan, and Howie Choset. Utility-guided palpation for locating tissue abnormalities. *IEEE Robotics and Automation Letters*, 2(2):864–871, 2017.

[8] Elif Ayvali, Hadi Salman, and Howie Choset. Ergodic coverage in constrained environments using stochastic trajectory optimization. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5204–5210. IEEE, 2017.

[9] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[10] Stanley J Benkoski, Michael G Monticino, and James R Weisinger. A survey of the search theory literature. *Naval Research Logistics (NRL)*, 38(4):469–494, 1991.

[11] Randal W Beard and Timothy W McLain. Multiple uav cooperative search under collision avoidance and limited range communication constraints. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 1, pages 25–30. IEEE, 2003.

[12] Matthew Flint, Emmanuel Fernandez-Gaucherand, and Marios Polycarpou. Cooperative control for uav's searching risky environments for targets. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 4, pages 3567–3572. IEEE, 2003.

[13] Marios M Polycarpou, Yanli Yang, and Kevin M Passino. A cooperative search framework for distributed agents. In *Proceeding of the 2001 IEEE International Symposium on Intelligent Control (ISIC'01)(Cat. No. 01CH37206)*, pages 1–6. IEEE, 2001.

[14] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[15] Jinwen Hu, Lihua Xie, Kai-Yew Lum, and Jun Xu. Multiagent information fusion and cooperative control in target search. *IEEE Transactions on Control Systems Technology*, 21(4):1223–1235, 2012.

[16] Masoud Dadgar, Shahram Jafari, and Ali Hamzeh. A pso-based multi-robot cooperation method for target searching in unknown environments. *Neurocomputing*, 177:62–74, 2016.

[17] Timothy H Chung and Joel W Burdick. Multi-agent probabilistic search in a sequential decision-theoretic framework. In *2008 IEEE International Conference on Robotics and Automation*, pages 146–151. IEEE, 2008.

[18] Tomonari Furukawa, Frederic Bourgault, Benjamin Lavis, and Hugh F Durrant-Whyte. Recursive bayesian search-and-tracking using coordinated uavs for lost targets. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2521–2526. IEEE, 2006.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[20] Guillaume Sartoretti, Yue Wu, William Paivine, TK Satish Kumar, Sven Koenig, and Howie Choset. Distributed reinforcement learning for multi-robot decentralized collective construction. In *Distributed Autonomous Robotic Systems*, pages 35–49. Springer, 2019.

[21] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[22] Juliani Arthur. Simple reinforcement learning with tensorflow part 8, dec 2017.

[23] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[24] Yongming Rao, Jiwen Lu, and Jie Zhou. Attention-aware deep reinforcement learning for video face recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 3931–3940, 2017.

[25] Dongbin Zhao, Yaran Chen, and Le Lv. Deep reinforcement learning with visual attention for vehicle classification. *IEEE Transactions on Cognitive and Developmental Systems*, 9(4):356–367, 2016.