Deep reinforcement learning based multi-agent pathfinding

Luo Zhiyao
A0209423L
e0452733@u.nus.edu
Multi-Agent Robotic Motion Laboratory
National University of Singapore

Guillaume Sartoretti
guillaume.sartoretti@nus.edu.sg
Multi-Agent Robotic Motion Laboratory
National University of Singapore

ABSTRACT

Multi-agent pathfinding (MAPF) is a crucial topic at the centre of many large-scale robotic applications from logistic distribution systems to simultaneous localization and mapping. recent works have proposed new AI-based approaches to MAPF, one of which called PIRMAL, casting MAPF into the reinforcement learning framework where agents are expected to learn full-decentralized policies under the demonstration of an expert system. This paper extends the previous work on 2D PRIMAL to 3D search space and discusses the communication mechanism of PRIMAL, as PRIMALc, in 2D search space. By careful reward shaping and gradient clipping, introducing communication into PRIMAL make imitation loss convergence steadily to a relatively low value.

KEYWORDS

Multi-agent System, pathfinding, Deep Reinforcement Learning, Imitation Learning

1 INTRODUCTION

Multi-agent pathfinding (MAPF) on grid maps is a challenging problem with numerous real-life applications such as surveillance [1], search and rescue [2], and warehouse distribution service [3]. To exploit the benefits of multi-agent pathfinding systems, in addition to solving all the challenges facing a single robot systems, scalability and path optimality are two of the most challenging aspects of MAPF. These two challenge are particularly prominent as the prevailing of drones, because MAPF expands the search space from 2D to 3D. On the other hand, agents are expected to communicate each other to plan paths in a more effective way. However, due to the stochasticity of environment, it is delicate to exchange useful information between agents.

In this paper, we focus on planning paths for a very large population of agents (e.g. one thousand) on a square grid map. To deal with the trade-off between path planning quality and computational cost, Pathfinding via Reinforcement and Imitation Multi-agent Learning (PRIMAL) [4] was recently proposed to provide a time-efficient, scalable and uncertainty-robust solution. By taking into account the positions of other agents, PRIMAL let agents favor policies that will benefit the whole team and not only themselves. Agents essentially learn a decentralized policy in this framework. They still exhibit implicit coordination during online planning by simultaneously learning single-agent planning via RL, and imitating a centralized expert path planning algorithm.

According to their experiments, PRIMAL can scale to various team sizes, world sizes, and obstacle densities, despite only providing agents local observation of the world. In low obstacle-density environments, PRIMAL exhibits tied performance, and even outperforms centralized MAPF planners which have access to the whole state of the environment.

Extending previous work of PRIMAL [4], the main contributions of this paper 1) introduce a conventional communication mechanism PRIMALc, and 2) expands the search space to 3D space. By implementing our communication mechanism, agents have access to others' predicted actions in the next few steps within their limited field of view. In order to fit this predicted actions into the PRIMAL network, communication between agents is represented in the form of prediction maps, which depicts the future positions of agents using binary matrices. Since the communication mechanism does not change the overall structure of PRIMAL, the communicating PRIMAL is still distributed and can be copied onto any numbers of agents. Experimentally, we discover that the implementation of the communication mechanism results in higher sensitivity to hyper-parameters, and longer training episodes to convergence as the prediction step grows comparing to original PRIMAL.

The paper is structured as follows: In Section 2 and Section 3 we review the fundamental trade-off of MAPF algorithms and summarize some state-of-the-art approaches and RL frameworks. In Section 4 we define the MAPF environment and present our assumptions for the methodology. Section 5 demonstrates how original PRIMAL is expanded from 2D space to 3D. In Section 6 we propose a variant of PRIMAL: PRIMALc, which allows agents to announce their future actions to other agents. Subsequently in Section 7, a series of tests are carried out to measure the performance of communicating PRIMAL comparing with original PRIMAL and we presents simulation results of PRIMALc. Section 8 concludes the remarks.

2 BACKGROUND

The biggest challenge of large-scale multi-agent pathfinding lies in the fundamental trade-off between path optimality and computational complexity. The trade-off is illustrated by the differences between centralized (or coupled) and decoupled algorithms to multi-robot path planning.

Coupled approaches, most of which are the generalization of single-agent planners in multi-agent cases, use the high-dimensional joint configuration space of a multi-robot system as the search space. Configuration space is the space of the complete specification of all the agent positions in the system. They can commonly find collision-free paths in minimal cost globally, but at high computational cost [5], [6]. Besides, Running a centralized, systematic path planning strategy such as A* [7] scales up poorly in practice, since both the search space and the branching factor grow exponentially in the number of agents.

Decentralized approaches, which decompose a problem into several sub-problems, can be faster and can work for larger problems [8]. Instead of searching the whole joint configuration space, decoupled algorithms explore a low dimensional space to compute a path for each agent individually; However, many existing decentralized methods offer no guarantees concerning the completeness, running time, and solution quality, which would bring about ill-performed path planning especially when agents are required to avoid colliding other agents in small environments [9], [10], [11], [12], [13].

3 RELATED WORK

3.1 M*-Subdimensional expansion

Subdimensional expansion is a framework to extend the scalability of coupled planner in MAPF cases where the configuration space of each robot is represented by a graph. First, a underlying coupled planner computes individual policy for each robot, specifying the individually optimal path from each point in the whole configuration space to the goal, neglecting the presence of other robots. When robots collide in the their individual policies, subdimensional expansion grows the dimensionality of the search space locally to perform collision avoidance using a coupled algorithm. Although the search space may grow to cover the entire joint configuration space in the worst case, subdimensional expansion can reduce the dimension of search space effectively for many problems.

As an implementation of subdimensional expansion, M* uses A* [14] as the underlying coupled planner. M* is proved to have the same optimality and completeness properties as A* [15].

In particular, OD-recursive-M*(ODrM*) [16] can further reduce the dimension of joint configuration space by breaking it down into independent collision sets, combined with Operator Decomposition(OD) [6] to keep the branching factor small during the search.

3.2 MAPF via Reinforcement Learning

Reinforcement learning, as an area of machine learning, concerns how to maximize the cumulative reward set by users by developing a good strategy (i.e., policy) based on experience through trial and error. Specifically, deep reinforcement learning has shown its potential to solve MAPF problem due to its environmental adaptability and flexibility [17], [18].

3.3 Advantage Actor-Critic(A2C)

Considering the Monte Carlo policy gradient strategy REINFORCE [19], the reward is calculated until the end of the episode, following the unbiased estimation of action-value function under the current policy at time t. The weight is subsequently updated by

$$\theta \leftarrow \theta + \alpha \nabla_\theta log\pi_\theta(s_t, a_t) v_t. \tag{1}$$

If a high reward $R(t)$ is produced, all actions that agents took are considered to be good, even if some are bad. This produces slow learning with large variance.

To improve the training speed of Monte Carlo strategy, and to expand it to non-episodic RL problems, the reward is required to update at each time step. Instead of using the total rewards, actor-critic approach [20] uses the approximation of the total reward $\hat{q}(s_t, a_t, w_t)$ which produces predictions of reward at each time step t. Therefore policy and value are updated by

$$\Delta\theta = \alpha \nabla_\theta (log\pi_\theta(s, a)) \hat{q}(s, a, w), \tag{2}$$

$$\Delta\sigma = \beta \left(R(s, a) + \gamma \hat{q}_\sigma(s', a')) - \hat{q}_\sigma(s, a)\right) \nabla_\sigma \hat{q}_\sigma(s, a), \tag{3}$$

where s' and a' denotes the state and action in time step $(t + 1)$.

Further, Advantage Actor-Critic eventually provides a good solution to measure the badness of an action by setting the state value as baseline. Advantages function is defined by $A(s, a) = Q(s, a) - V(s)$, telling the improvement compared to the average performance of all actions at the current state. In practice, TD error can be a good estimator of the advantage function. Then the advantage function is obtained by

$$A(s, a) = r + \gamma V(s') - V(s). \tag{4}$$

In section 5.4, we will demonstrate how the advantage function helps to convergence our network in the asynchronous manner.

4 ASSUMPTIONS

4.1 Search Space

In this paper, the MAPF problem is discussed in discrete space (2D or 3D), where the environment can be represented by a grid world. An agent, an obstacle and a goal position occupy respectively a unit grid (1 by 1 in 2D space, 1 by 1 by 1 in 3D space) in the environment. An example of a discrete MAPF environment is shown in Fig.1.

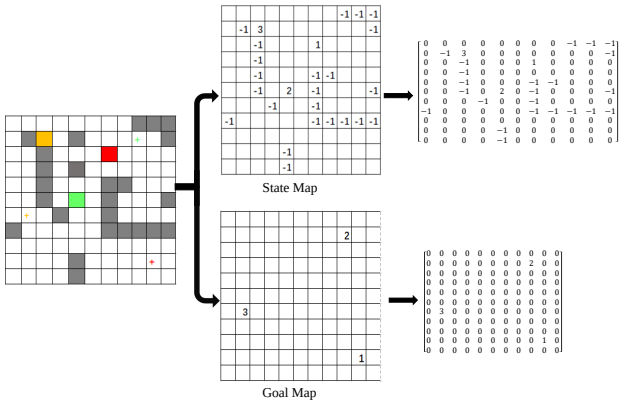

Figure 1: 2D Representation of State and Goal Maps

We assume that agents can move in the 4-connected region. In 2D space, considering an agent located at position (x_0, y_0), it has maximum 5 valid actions (up, down, left, fight, and standing still), which would result in the state transition to (x_0, y_0+1), (x_0, y_0-1), (x_0-1 y_0), (x_0+1, y_0), (x_0 y_0) correspondingly. Whilst in 3D space, including standing still, an agent has maximum 7 valid actions along with both positive and negative directions of axis x, y and z, which will respectively transit the original state (x_0, y_0, z_0) to (x_0, y_0, z_0), (x_0+1, y_0, z_0), (x_0-1, y_0, z_0), (x_0, y_0+1 z_0), (x_0, y_0-1, z_0), (x_0, y_0, z_0+1) and (x_0, y_0, z_0-1). We also assume that all the agents move at the same speed, i.e., 1 unit cell per time step.

4.2 Collision

Collisions between agents occur when two agents simultaneously occupy the same location or simultaneously cross the same edge in opposite directions. It is also possible that agents collide with obstacles, or hit the boundary of the environment. When the collision occurs in the above-mentioned conditions, the colliding agents will be forced to remain as its current state instead of taking its action. Any action which will result in a collision next step is denoted as an invalid action. Fig.2 demonstrates four particular cases where agents are allowed or prevented to perform their actions.

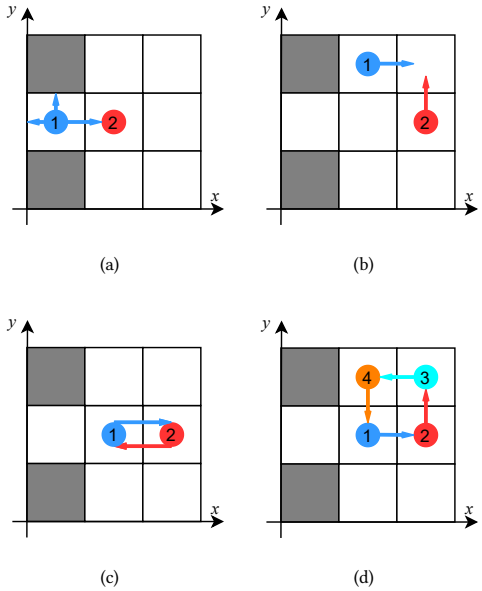

Figure 2: Collision Checking: (a) Agent collides with wall/boundary/another agent (b) Two agent collide when reaching a new state simultaneously. (c) Two agent collide when swapping positions. (d) Agents move in circle. This is allowed in the setting of most coupled planners, but is infeasible in our approach.

4.3 Simulation World

The world map is a two-layered matrix stored in a centralized environment. The first layer is called state map, where agents are denoted by its *agentID*, obstacles are denoted by -1, free space denoted by 0, and the second layer is the goals map, where goals are denoted by the *agentID* it belongs to.

4.4 Realizing MAPF Environment as A Reinforcement Learning Problem in Python

Since the MAPF environment we define is discrete, we can represent the entire state space as a double-layer matrix containing the state and goal maps, which has been discussed in Section 4.1. The environment is centralized, with a set of functions responsible for the interactions with agents. The environment is initialized with a world map that contains all the state information of agents, obstacles, and goals. At the start of a time step, the environment assigns observation arrays to all the agents according to the current states of the agents. Then agents make their decisions to determine where they would go at the next time step based on the observation received from the environment. The actions that agents make will correspondingly be sent to verify validation by the environment. The environment will automatically replace any invalid actions that agents may decide to take with standing still, and the corresponding penalty on agents' rewards will be dealt. After proceeding with all the actions of the agents, the environment finally allows agents to transit to the next state. The pseudo-code of this centralized environment is shown in Algorithm 1.

Algorithm 1 Decentralized MAPF System In A Centralized Environment

```
function MAPFEnv
    Initialize world map
    for episode = 0 → max_episode do
        for step = 0 → max_step do
            for agentID = 1 → num_agents do
                observation ←Observe(agentID)
                action_dist ←Local_Net(observation)
                action ←Choice_Action(action_dist)
                Step(agent, action)
            end for
            update loss
            update reward
        end for
    end for
end function
```

5 3D REPRESENTATION OF PRIMAL

In this section, we present how the MAPF problem is fitted into the reinforcement learning framework in 3D space. We demonstrate the observation and action space of each agent, the reward structure and the neural network that represents the policy to be learned comparing with 2D PRIMAL.

5.1 Observation Space

Considering a partially-observable discrete grid world where agents can only observe the state of the world in a 11 by 11 field of view centered around themselves, the observation of each agent consists of 4 layers of sub-observation respectively containing obstacle position, neighbor agents' positions, neighbors' goals and agent's goal to simplify the learning task. All four sub-observations are binary. When agents are close to the edges of the world, obstacles are added at all positions outside the world's boundaries.

We believe that partial observation is critical to PRIMAL especially in the 3D case. Since the dimension of observation expands from 3 to 4 ($4 \times 11 \times 11$ to $4 \times 11 \times 11 \times 11$) while obstacle density remains unchanged, the observation information would become highly sparse. To decrease the input dimension and reduce the possibility of divergence, setting observation size equal to $11 \times 11 \times 11$

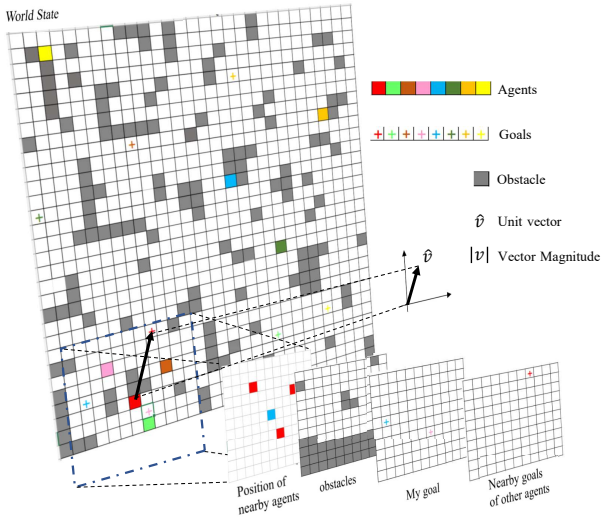

Figure 3: Observation of a 2D MAPF Environment. Agents are displayed as colored squares, their goals as similarity-colored crosses, and obstacles as gray squares. Each agent only has access to a limited field of view centered around its position (in this case 11 by 11), where observation information is broken down into 4 channels: positions of obstacles, position of nearby agents, goal positions of these nearby agents, and position of its goal if within the field of view. Positions that are outside of the world's boundary is considered to be obstacles.

is effective experimentally. Besides, each agent needs to have access to information about its goal, which is often outside of its field of view in partial-observation case. We give each agent both a unit vector pointing towards its goal and the Euclidean distance to its goal as part of their observation data, which is shown in 3.

5.2 Action Space

Agents take discrete actions in the grid world: moving one cell in one of the six cardinal directions in 3D space (forward, back, up, down, left, right) or staying still. During training, actions are sampled only from valid actions and additional loss functions aids in learning this information. In other words, all the invalid actions will be filtered by the environment in the training process. This will experimentally enable more stable training, compared to giving punishment for selecting invalid moves.

If an agent selects an invalid move during testing, it instead stands still for that time step, and the corresponding penalty is dealt. In our tests, agents very scarcely perform invalid moves once fully trained, showing that agents sufficiently learn the set of valid actions in each state.

Additionally, to combat convergence to back-and-forth moving strategies, agents are prevented during training from returning to the location they occupied at the last time step, but they are allowed to stay still during multiple successive time steps, which is to guarantee agents a way to turn back or allow agents to stay on their goals once reached. This is necessary to encourage exploration and learn effective policies even during M* imitation learning process.

5.3 Reward

We design our reward function following a similar structure of RL settings in MAPF. As shown in Table 1, agents are punished (i.e., -0.3) for each time step they are not stepping on goal in order to encourage them to reach their goals as soon as possible. We set the staying still penalty (i.e., -0.5) to be slightly more than the moving penalty to encourage exploration. Collisions (no matter with walls, boundaries, or agents) lead to a 2 penalty. Agents receive a +20 reward for finishing an episode if and only if all agents step on their goals simultaneously.

Table 1: Reward Setting

Action	Reward
Move [N/E/S/W]	-0.3
Agent Collision	-2.0
No Movement(on/off goal)	0.0 / -0.5
Finish Episode	+20.0

5.4 Network Structure

Our work rely on the asynchronous advantage actor-critic (A3C) algorithm [21], following the same network structure as PRIMAL. For the network of each agent, a deep convolutional network is designed to extract features from the observation it receives. We use the same 6-layer convolutional network structure pictured in Fig.4 as implemented in the definition of original PRIMAL. The only changes is that 2D convolutional kernels are replaced with 3D kernels to process the 3D observation.

During training, the policy, value, and "blocking" outputs are updated in a batch when an episode finishes or exceeds maximum episode length (256 in our case). The value is updated to match the total discounted return $R_r = \sum_{i=0}^{k} \gamma^i r_{t+i}$. The policy loss is given by

$$L_\pi = \sigma_H \cdot H(\pi(o)) - \sum_{i=1}^{T} log\left(P\left(a_t|\pi, o; \theta\right) A\left(o_t, a_t; \theta\right)\right)), \quad (5)$$

where $H(\pi(o))$ is the entropy term to the policy loss, σ_H a small entropy weights, and $A(o_t, a_t; \theta)$ the value function. The entropy term is obtained by

$$H(\pi(o)) = -p(a_i) \cdot \sum_{i=1}^{n} f_t(log(p(x_i))), \quad (6)$$

where $p(a_i)$ denotes the probability distribution of unclassified information. In our case $p(x_i)$ represents the confidence of each action (in 3D space 7 actions). It is shown that adding the entropy term encourages exploration and discourage premature convergence [22]. And we use an approximation of the advantage function by bootstrapping using the value function denoted by

$$A(o_t, a_t; \theta) = \sum_{i=1}^{n} \gamma^i r_{t+1} + \gamma^k V(o_{k+1;\theta} - V(o_t; \theta)). \quad (7)$$

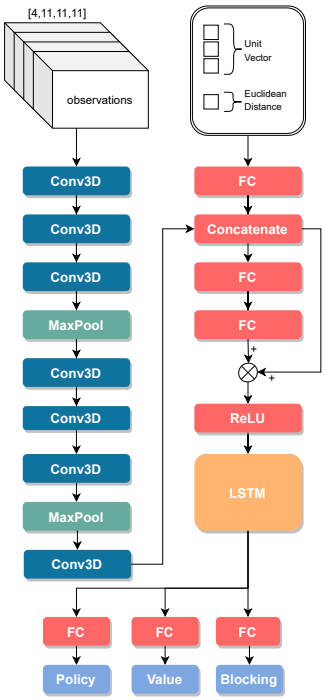

Figure 4: Network Structure of 3D PRIMAL

Besides the policy loss, we need two additional loss functions L_{block}, the log likelihood of predicting incorrectly, and L_{valid}, the log likelihood of selecting an invalid move [23]. These two losses are represented by

$$L_{valid} = \sum_{i=1}^{n} log(v_i) \cdot p_v(i) + log(1 - v_i) \cdot (1 - p_v(i)), \quad (8)$$

$$L_{block} = \sum_{i=1}^{n} log(b_i)) \cdot p_b(i) + log(1 - b_i) \cdot (1 - p_b(i)), \quad (9)$$

$$v(i) = Sigmoid(p_a(i)) \qquad i = 1, 2, ..., 7, \quad (10)$$

where $p_v(i)$ and $p_b(i)$ denote the confidence of action validation and blocking predicted by agent i, v_i and b_i respectively denote the ground truth of the action validation and blocking status provided by the environment.

6 DECENTRALIZED MAPF WITH LOCAL COMMUNICATIONS

Recall that we implement PRIMAL to decentralize the decision-making behavior of M*, the coupled, optimal and complete solution to MAPF knowing the global configuration space. However, M* works under the assumption that other agents are regarded as moving obstacles, which means an agent cannot have access to where other agents will go at the next time step. This section discusses our proposed communication mechanism, which allows agents to know others' future actions in a particular way.

6.1 PRIMALc

Agents are not able to foresee others' future actions intuitively due to causality constrain. However, instead of providing the true future actions (i.e., obtained by taking one step ahead of the current time step and tracing back), we enable the agent to have access to the prediction actions generated by other agents. In this case, the prediction step represents the action a certain agent would like to take if nobody else would move in the current time step. The prediction actions will then perform as an additional input into the neural network, which opens a new channel for agents to make a better decision taking the advantage of others' prediction steps where the remaining agents are considered as static obstacles. To perform the prediction step for each agent, we also modify the observation (i.e., input structure of the network) and network structure correspondingly.

6.2 Observation and Network Modification

Recall that in original PRIMAL the observation of each agent consists of 4 layers of binary matrices (2-dimensional or 3-dimensional matrix depending on 2D or 3D search space). Communication of PRIMAL adds a set of binary matrices that illustrate the predicted actions of other agents. We call these additional sub-observation layers 'prediction maps'. Comparing to real time step, we denote time steps in the prediction process as 'prediction steps'. The number of prediction steps is set by users.

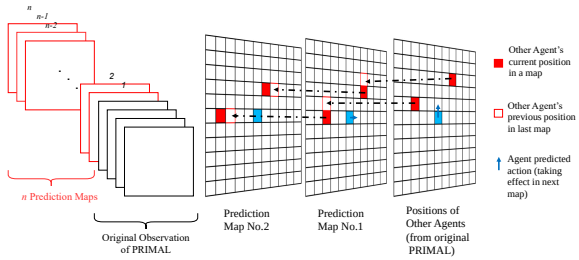

Figure 5: Observation of PRIMAL with communication where $Pred\_step = n$. Taking $Pred\_step = 2$ as an example, other agents are assumed to stand still in prediction maps No.1 and No.2, only blue agent can move.

The number of layers of the prediction maps is consistent with predicted step, representing how many future steps ahead agents will let others know. Similar to the agent's position map (as shown in fig 3), prediction maps show the relative predicted positions of agents in the limited field of view. An example of observation representation is shown in fig 5. Therefore, the observation includes $4 + n$ layers of binary matrices, where n is the number of predicted steps defined by users.

6.3 Training of PRIMALc

Different from the original PRIMAL, PRIMAL communication is trained not only by current observation (i.e., 4 layers matrices same as original PRIMAL) but also trained by future observation (i.e., prediction maps). To generate prediction maps, the predicted actions of all agents are desired. Thus, agents will perform their

predicted actions in the advance of interacting directly with the environment in an iterative way.

Algorithm 2 Centralized Implementation of PRIMALc

```
function Compute_Future_Actions
    Create a mirror world
    for prediction_step = 0 → max_pre_step do
        current_ob ←Observe(current_state)
        pred_map ←Observe(others_action)
        pred_action ←AC_Net([current_ob, pred_maps])
        upload pred_action to environment
        wait for other agents to upload
        update loss
        update reward
        pred_map ← Pull_From_Env(agentID)
        pred_map_list .append(pred_map)
    end for
    return (pred_map_list)
end function

function Main_Training_Loop
    Create a mirror world
    for each real_time_step do
        for each agentID do
            action ←AC_Net([current_ob, pred_maps])
            pred_maps ←Compute_Future_Actions(current_s)
            current_ob ←Observe(current_s)
            current_s ←Step(action)
        end for each
    end for each
end function
```

Prediction maps are initialized by copying the neighbor map for n times, where n is the number of prediction steps. Before determining the next action, agents will first switch to 'prediction mode'. In this mode, each agent uses its local network to generate their preferred n future actions, assuming no other agents moves in the system. After all the agents have created their predicted actions, they will broadcast those actions to their neighbors within their field of views. They subsequently make their prediction maps based on others' announcement of predicted actions. Then, 'prediction mode' is turned off. Until all prediction maps are created, agents determine their real next-step action using the prediction maps. Algorithm 2 shows the pseudo-code of this mechanism.

Recall that in section 5.2 we sample real actions only from valid actions for original PRIMAL. In communicating PRIMAL, however, we also need to consider how to sample predicted actions of agents besides their real actions. For the training result we present, both real agent actions and their predicted actions are sampled from all valid actions with their respective possibilities, i.e., Boltzmann action selection.

In experiments we found out that losses and reward curves vibrate much more than original PRIMAL during the training, depending on the value of *prediction_step*. The vibration generally leads to slower training efficiency and sometimes divergence. A fine-tuned gradient clipping value can help to contain the vibration to some extends but cannot completely attenuate it. Especially, when the prediction step is set to a relatively large value, (e.g. larger than half of the observation size), agents are not able to produce accurate individual predicted actions since their current network have no access to the information out of their field of views. Following on upon with the inaccurate or even wrong prediction maps, correct decisions are difficult to be made.

We are also trying to investigate the influence if *pred_step* is set larger than the boundary of agents' field of views. We subsequently develop a series of measurement to improve the performance of communication, e.g. using exploitation strategy (i.e., always select the most confident action) during the prediction process. Due to the limitation of computational resource, however, we only finish the training and testing when *pred_step* is equal to 2 and implementing Boltzmann action selection until the finalization of this report. A set of half-trained experiments are shown in 6. In Section 7 we will make discussion exclusively based on the training result we have fully obtained.

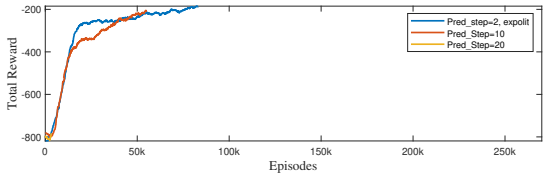

Figure 6: Reward plotting of Unfinished training

7 TRAINING RESULT WITH COMMUNICATION AND SIMULATION

7.1 Training Result of PRIMALc

According to our testing, PRIMALc exhibits on-par performance with original PRIMAL when $Pred\_step = 2$. Whether PRIMALc would outperform original PRIMAL when *Pred_step* is set to be a larger value still requires more investigation, but we believe that for larger values of n, the added information being shared between agents, if used correctly, should significantly improve over the original PRIMAL.

The implementation of our communication mechanism converges the imitation loss. As shown in Fig. 7, imitation loss of communicating PRIMAL goes down to a low value as training episodes grow, while the curve of original PRIMAL vibrates at 1.2. Although the imitation loss becomes sufficiently small, communicating PRIMAL and original PRIMAL are shown to achieve on-par performances on the episodes length during training. We conjecture that the imitation loss (i.e., cross entropy loss between expert's actions and agents' actions) cannot fully measure the proximity from agents' decisions to expert's decisions, which may also be an evidence of why original PRIMAL outperforms pure RL even if the imitation loss does not go down. Due to time limitation, testing is not finished by the deadline.

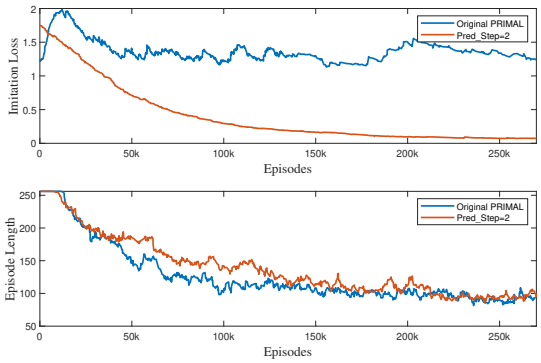

Figure 7: Training Comparison Between Communicating PRIMAL and Original PRIMAL

7.2 Simulation

To implement our algorithms in real application such as warehouse distribution, it is inevitable to transfer our discrete environment into a continuous space. Thus, We set up a set of simulation in Airsim, a simulator for drones built on Unreal Engine. We manually mesh the continuous space of UE environment into grids with 200 centimeter length/width/height.

Airsim APIs can automatically find a smooth trajectory between two adjacent points in the continuous space following embedded PID control so that we can visualize the pathfinding process by giving the discrete paths which are generated by our trained network. Experimentally we set the average velocity of drones to be $1m/s$ to avoid excessive position overshoots. The video can be found at https://youtu.be/8gqtUsSnnjg.

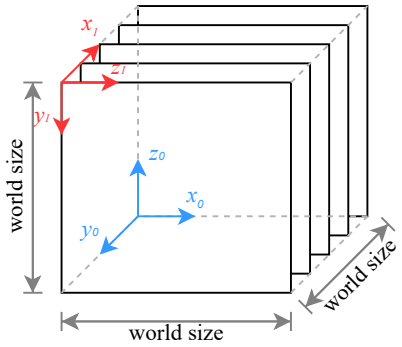

Figure 8: Homogeneous Transformation From M^* Coordinate to Simulation Coordinate

Since Od_M * use different coordinate, We transform the coordinate frames following

$$P_a = A_x(-\frac{\pi}{2})A_z(-\frac{\pi}{2})T(0,-l,-l)\times P_{M^*} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -l \\ 1 & 0 & 0 & -l \\ 0 & 0 & 1 & 1 \end{bmatrix} \times P_{M^*} \tag{11}$$

where T_{M^*} denotes the frame in M^* representation, T_a denotes standard Cartesian frame in UE environment. To transform the path generated by M^* algorithm, each path point represented in UE frame is derived by $P' = T_a \times P$, where P' and P are respectively the augmented vectors in frame a and frame M^*.

By our exclusive simulation, drones can move smoothly along the discrete path point controlled by built-in controller without any collisions. In this manner, we dramatically simplify the path planning problem, because drones only concern about high-level commands (i.e., positions given by our approaches) rather than complex dynamics and kinematics.

8 CONCLUSION

In this paper, we expand PRIMAL to 3D search space and propose a variant of PRIMAL, PRIMALc, which allows agents to communicate by exchanging their predicted future actions with each other. Our experiments indicate that communicating PRIMAL needs 10% more episodes to converge comparing with original PRIMAL. It also brings training process higher sensitivity towards hyperparameters as the *pred_step* grows.

Future work will focus on the improvement of communicating PRIMAL, including designing a prediction loss to slightly punish the use of wrong predicted action, modifying action selection approaches, etc. As the dimension of the input tensor (i.e., observation matrices) expands with the number of prediction step, the difficulty of feature extraction grows as well. Thus, the insight is to develop a more effective network to replace the 9 layers of CNN structure in original PRIMAL.

Permitting agents to change their velocity provides another perspective direction. Intuitively, agents would be able to perform more flexible collision avoidance strategies if they are allowed to shift their speeds in the cases that an agent has to stand still to let other agents pass first. Future work is expected to discover the influence of discrete velocity towards multi-agent pathfinding problems. To prevent agents from maximizing their speeds all the time, an accelerate penalty should be dealt to the speeding-up agents. Besides, agents which run at a higher speed may have larger punishment when they collide.

REFERENCES

[1] Paolo Remagnino, Tieniu Tan, and Keith Baker. Multi-agent visual surveillance of dynamic scenes. *Image and Vision Computing*, 16(8):529–532, 1998.
[2] Jean Berger and Nassirou Lo. An innovative multi-agent search-and-rescue path planning approach. *Computers & Operations Research*, 53:24–31, 2015.
[3] Kevin Nagorny, Armando Walter Colombo, and Uwe Schmidtmann. A service- and multi-agent-oriented manufacturing automation architecture: An iec 62264 level 2 compliant implementation. *Computers in Industry*, 63(8):813–823, 2012.
[4] 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.
[5] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
[6] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.
[7] František DuchoĖ, Andrej Babineca, Martin Kajana, Peter BeĖoa, Martin Floreka, Tomáš Ficoa, and Ladislav Jurišicaa. Path planning with modified a star algorithm for a mobile robot. *Procedia Engineering*, 96:59–69, 2014.
[8] Vishnu R Desaraju and Jonathan P How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.

[9] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1-4):477, 1987.

[10] Kamal Kant and Steven W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The international journal of robotics research*, 5(3):72–89, 1986.

[11] Stephane Leroy, Jean-Paul Laumond, and Thierry Siméon. Multiple path coordination for mobile robots: A geometric algorithm. In *IJCAI*, volume 99, pages 1118–1123, 1999.

[12] Mitul Saha and Pekka Isto. Multi-robot motion planning by incremental coordination. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5960–5963. IEEE, 2006.

[13] David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[14] Florian Grenouilleau, Willem-Jan van Hoeve, and John N Hooker. A multi-label a* algorithm for multi-agent pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 181–185, 2019.

[15] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

[16] Cornelia Ferner, Glenn Wagner, and Howie Choset. Odrm* optimal multirobot path planning in low dimensional search spaces. In *2013 IEEE International Conference on Robotics and Automation*, pages 3854–3859. IEEE, 2013.

[17] Sachiyo Arai, Katia Sycara, and Terry R Payne. Multi-agent reinforcement learning for planning and scheduling multiple goals. In *Proceedings Fourth International Conference on MultiAgent Systems*, pages 359–360. IEEE, 2000.

[18] Gang Lei, Min-zhou Dong, Tao Xu, and Liang Wang. Multi-agent path planning for unmanned aerial vehicle based on threats analysis. In *2011 3rd International Workshop on Intelligent Systems and Applications*, pages 1–4. IEEE, 2011.

[19] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[20] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[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] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

[23] 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.