MARL with Local Observation & Communication

Georg Ye
georye@student.ethz.ch

Owen Du
owen.du@student.ethz.ch

Leander Diaz-Bone
ldiazbone@student.ethz.ch

Abstract

Multi Agent Reinforcement Learning (MARL) extends Reinforcement Learning to multiple agents interacting with the underlying Markov Decision Process (MDP) and optimizing their own rewards. MARL has applications in areas such as self-driving cars and drone swarms. Many previous algorithms assume global observability and centralized coordination, which are unrealistic for many applications. To address this, we introduce the MARL with Local Observation & Communication setting and develop the IPPO with Belief algorithm tailored for this scenario. We evaluate our method on two environments, comparing it with state-of-the-art algorithms. Our results suggest that our method is competitive with those using global observability and communication in many cases.

1 Introduction

Multi-Agent Reinforcement Learning (MARL) extends Reinforcement Learning to multiple agents interacting with the underlying MDP to optimize their rewards [1]. This extension is crucial for real-world applications like self-driving cars and drone swarms, where coordination is essential for optimal behavior and safety. Many existing algorithms assume global observability and centralized coordination [1], which are impractical in many scenarios. For example, in a fleet of autonomous cars, communication with a central server for decision-making introduces significant overhead and risks during connectivity issues. Similarly, local car behavior is influenced mainly by nearby cars, making global observation and communication unnecessary. This example demonstrates the need for localized observation, communication, and planning.

To address these issues, we introduce the MARL with Local Observation & Communication setting. In this framework, agents have a local view of the global state and can only exchange information if they are sufficiently close. This setting will be formally defined in the Methods section in Defintion 1. The main challenge is developing an efficient algorithm in this more restrictive environment. The MARL setting is inherently challenging due to non-stationarity and a large state-action space; further restrictions on observability and communication exacerbate this difficulty. Therefore, our goal is to develop an algorithm capable of learning good policies in simple problems rather than establishing theoretical guarantees.

1.1 Related Work

There is extensive literature on MARL using various models for communication, observation, and planning. The authors of [9] proposed the Networked Multi-Agent MDP, which uses time-varying networks to limit communication to adjacent agents. A similar formulation will be used to introduce local communication in our algorithm. The authors propose two algorithms and establish convergence guarantees for this setting. Although, the communication in the Networked Multi-Agent MDP is decentralized, the authors assume a fully observable states and actions. Lowe et al. proposed the

Foundations of Reinforcement Learning, FS2024. Department of Computer Science, ETH Zürich.

MADDPG algorithm, focussing on decentralized execution [6], showing effective learning in complex environments. Their algorithm is based on the actor-critic framework, which will also serve as the basis for our method. However, MADDPG's reliance on a central critic is not feasible in a local communication setting.

We will outline three additional MARL algorithms, which we use as reference algorithms in our experiments, along with their respective assumptions.

2 Methods

This section first formally defines locality in observation and communication. Next, we present three reference algorithms used as building blocks and comparisons for our algorithm in experiments. Furthermore, we introduce the IPPO with Belief algorithm.

2.1 Multi-Agent MDP with Local Observation & Communication

Definition 1 *(Multi-Agent MDP with Local Observation & Communication)*
We define a Multi-Agent MDP with Local Observation & Communication as a tuple $(\mathcal{S}, \{\mathcal{A}_i\}_{i\in[N]}, \mathcal{P}, \{\mathcal{R}_i\}_{i\in[N]}, \{\mathcal{C}_t\}_{t\geqslant 0}, \{O_i\}_{i\in[N]}, \{o_i\}_{i\in[N]})$. *$\mathcal{S}$ denotes the global state space, $\mathcal{A}_i$ is the action space of agent i ($\mathcal{A} = \times_{i\in[N]} \mathcal{A}_i$), $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ denotes the transition kernel, and $R_i : S \to \mathbb{R}$ is the local reward function. Each $\mathcal{C}_t$ denotes a partition of $[N]$, where the elements $c_{t,b} \in \mathcal{C}_t$ denote sets of indices of the agents that are locally adjacent and can communicate. O_i is the set of local observations of agent i, and $o_i : S \to O_i$ is the function mapping the global state to a local oberservation for the agent i.*

In this setting, each agent has a truly local view of the environment, receiving only local observations o_i of the global state, seeing only his own rewards and actions, and communicating only with locally adjacent agents. We impose no constrictions on the communication between agents as long as agents can communicate. This definition of locality makes the setting more applicable to real-world scenarios. We want to highlight that the local observation is a deterministic mapping from the global state to local observations, unlike the commonly used partial observation, which maps to a probability distribution over observations. Figure 1 visualizes the notion of locality.

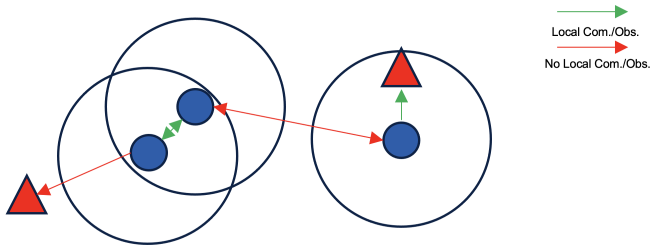

Figure 1: Visualization of Locality. The black circles around each agent (blue circles), visualize the observation and communication radius. Green arrows denote Local Observation/Communication, while red arrows indicate that there is none.

2.2 (I)PPO

Independent Proximal Policy Optimization (IPPO) [3] is an extension of the Proximal Policy Optimization (PPO) algorithm tailored for multi-agent systems with local observations. PPO, originally proposed by Schulman et al. [4], is renowned for its stable and efficient policy optimization in single-agent settings. Its robustness and effectiveness make it a suitable candidate for adaptation to multi-agent reinforcement learning (MARL).

In a multi-agent setting, the combined action space grows exponentially as the action space per agent is raised to the power of the number of agents, i.e., $actions\_per\_agent^{\#agents}$. PPO assumes access to a global state and outputs a single action, which is then mapped to individual actions for each agent.

However, the large action space in multi-agent settings could cause complications if it becomes too large.

Unlike standard PPO, which assumes a global state, IPPO leverages the local observations of each agent to make decisions. Each agent operates independently, updating its policy based solely on local observations and rewards, making it suitable for decentralized environments where global state information is inaccessible or impractical to use.

IPPO modifies the standard PPO approach by focusing on local observations (o_i) for each agent (i). The algorithm optimizes the following objective:

$$L^{\text{IPPO}}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)} \hat{A}_t, \text{clip} \left(\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \right]$$

where π_θ is the policy parameterized by θ, $\pi_{\theta_{\text{old}}}$ is the old policy, $\hat{A}_t$ is the advantage estimate, and ϵ is a hyperparameter for clipping (we set it to 0.2).

After some testing, we settled on the following architecture: two hidden layers, each of size 128, and Tanh as the activation function. The reasoning for these choices can be found in the appendix A.4. Additionally, we added an entropy term to the loss to promote exploration with a coefficient of 0.1.

2.3 IPPO with Belief

We propose the IPPO with Belief algorithm 1 for the Multi-Agent MDP with Local Observation & Communication setting 1.

Algorithm 1 IPPO with Belief

Input: θ_i^0 (belief state), ω_i^0 (actor par.), α_i^0 (critic par.), ϵ (clipping), γ (discount)
for $e = 0, ...$ **do**
  $R_i = [\,]\ \forall i \in [N]$ ▷ Initialize empty Replay Buffer
  for $t = 0, \ldots, T$ **do**
    for $i \in [N]$ **do**
      Observe o_i^{t+1}, r_i^{t+1} ▷ Local Obervation & Reward
      $\theta_i^{t+1} \leftarrow \arg\max_{\theta \in \Theta}\ q_\theta(s|\theta_i^t, o_i^{t+1})$
      $a_i^{t+1} \sim \pi_{\omega_i^e}(a|\theta_i^{t+1})$ ▷ Sample & Execute Action
      $R_i \leftarrow R_i \cup [(\theta_i^{t+1}, a_i^{t+1}, r_i^{t+1}, \pi_{\omega_i^e}(a|\theta_i^{t+1}), V_{\alpha_i^e}(\theta_i^{t+1}))]$
    end for
    for $c_{t,b} \in \mathcal{C}_t$ **do** ▷ Local Communication
      $q_{\theta_i^{t+1}}(s) \leftarrow \frac{1}{Z} \prod_{i \in c_{t,b}} q_{\theta_i^{t+1}}(s)\ \forall i \in c_{t,b}$
    end for
  end for
  for $i \in [N]$ **do**
    for $t = 0, \ldots, T$ **do** ▷ Compute Advantages
      $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_i^k$
      $A_t \leftarrow G_t - V_{\alpha_i^e}(\theta_i^t)$
    end for
    $\omega_i^{e+1} \leftarrow \arg\max_\omega \frac{1}{T} \sum_{t=0}^{T} \min(\frac{\pi_\omega(a|\theta_i^{t+1})}{\pi_{\omega_i^e}(a|\theta_i^{t+1})} A_t, \text{clip}(\frac{\pi_\omega(a|\theta_i^{t+1})}{\pi_{\omega_i^e}(a|\theta_i^{t+1})}, 1-\epsilon, 1+\epsilon) A_t)$
    $\alpha_i^{e+1} \leftarrow \arg\min_\alpha \sum_{t=0}^{T} (V_\alpha(\theta_i^t) - G_t)^2$ ▷ Optimization using Adam
  end for
end for

The IPPO with Belief algorithm builds on the IPPO algorithm 2.2. by estimating the global state using the belief state distribution $q_\theta(s)$ instead of using the local state directly. Local observation o_i^t updates the belief state distribution, and during communication, agents merge belief distributions with nearby agents. This effectively distributes information available at one agent to others, ensuring accurate estimation of the global state. Actor and critic updates resemble those in PPO. The hyperparameters

for the experiments are the same as those described in the (I)PPO section 2.2. Implementation details of the belief state distribution for different environemnts are in the Appendix A.2.1.

2.4 MAPPO

Multi-Agent Proximal Policy Optimization (MAPPO) [2] extends the Proximal Policy Optimization (PPO) algorithm to multi-agent environments by incorporating a centralized training with decentralized execution approach. In MAPPO, each agent learns its policy using both local and global information during training, enabling better coordination and performance in multi-agent settings.

MAPPO leverages a centralized critic that takes the global state as input to estimate the value function, while each agent maintains its decentralized actor, which uses local observations to select actions. This centralized training with decentralized execution allows the critic to use richer information to compute more accurate value estimates, thereby improving the actor's policy updates.

In our implementation, we did not use recurrent neural networks (RNNs) as suggested in some original MAPPO papers. Instead, we employed standard multi-layer perceptrons (MLPs). Both the actor and critic networks used one hidden layer with 64 units. Additionally, we included an entropy term in the loss function with a coefficient of 0.1 to encourage exploration. The hyperparameters used in our experiments included a hidden layer size of 64, one hidden layer for both the actor and critic, an entropy coefficient of 0.1, and a clipping parameter of 0.2.

3 Evaluation

This section discusses the environments used for the experiments and the rationale for choosing them. Furthermore, we propose different experiments, which will be used to evaluate the effectiveness of our method compared to the reference algorithms.

3.1 Lumberjack Environment

The Lumberjacks environment [5] is a cooperative MARL environment, where agents aim to cut down trees on a discrete 2D grid. Agents can move within the grid in each step and cut down trees if the number of agents is larger than or equal the tree strength of a particular tree. Thus, in order to perform well on this environment, the agents need to cooperate to clear the entire grid.

The global state of the Lumberjacks environment contains both an agent grid and a tree grid. However, only the local observations are exposed to the user, which contain for each agent local grids of agents and trees respectively. The default size for the local grid is 3x3 around the position of each agent. Additionally, for each agent the agent ID, position and number of steps taken are given in the local observation. Each agent is allowed to move to one of 4 adjacent grids (no diagonals) or not move at all, resulting in an action space that contains an integer from 0 to 4 for each agent. In order for the agents to learn to cooperate, they get a reward of 10 for each tree cut, but also incur a penalty of -1 for each step taken.

This environment requires the agents to cooperate, as a single agent cannot take down trees of higher strength. Additionally, due to the step penalty the agents try to find policies which clear the trees as fast as possible as opposed to moving in sync in every step.

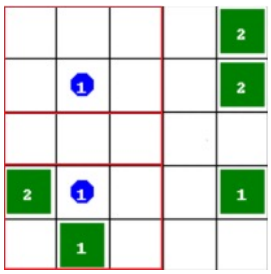

Figure 2: Lumberjacks Environment Visualization. The red boundaries indicate the observation radius of the agents, blue circles denote the agents and green squares denote trees, where the number inside the symbols represent the number of agents/trees on the specific field.

3.2 Cat & Mouse Environment

To benchmark the algorithms on another cooperative MARL environment, we propose the Cat & Mouse environment. The objective of the agents (cats) is to catch moving prey (mice). We develop both a discrete version where agents and prey lie on a grid and a continuous version, where actions and state are continuous variables. In the following part, we focus on the discrete environment to provide a better comparison to Lumberjacks, details about the continuous environment can be found in A.2.2.

In the discrete case, the environment is similar to the Lumberjacks environment. Cats and mice are located on a discrete grid, which also represents the global state of the environment. A cat catches all mice on a grid cell if they are in the same position at the end of a step. As in Lumberjacks, the local observation contains a local grid of agents and mice with position and ID of the current agent. We also adopt the rewards from Lumerjacks, where we reward all agents that participate in catching the prey and incur a penalty for each step taken. For the action space however, we extend the possible actions per agent to 9 to allow for diagonal steps.

The Cat & Mouse environments are implemented according to the Gymnasium [8] API. In the discrete environment, initially N agents and M prey are randomly placed on the grid. In each step, each mouse first detects agents within observation range. Then, it selects a random agent out of those to run away from. If the mouse is not on the boundary, it takes a single step in the opposite direction of the chosen agent. If the mouse is on the boundary, we calculate the possible grid cells the mouse can move to to increase its Manhattan distance from the chosen agent. Then, the mouse randomly chooses on of the possible directions. This means, that if 2 mice are on the same grid cell, they are able to split up in the next step. This way, the mice have random elements in their movement which further encourages cooperation between agent to catch the prey.

We choose a catch reward of 10 per mouse per agent, and a -1 step penalty for each agent. We also experimented with reward normalization, however, we did not obtain better results with this approach.

The Cat & Mouse environment setting require cooperation between agents, as the mice are coded to evade the agent within the observation radius. Thus, multiple agents need to cooperate to chase the prey to each other and catch them together.

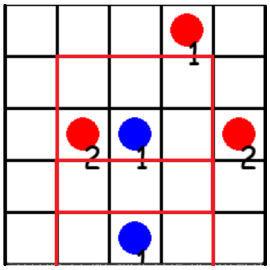

Figure 3: Discrete Cat & Mouse Environment Visualization. The blue circles represent the agents (or cats) and red circles represent the prey (or mice). The red boundaries denote the observation range of the agent.

3.3 Experiments

We conducted three experiments in the Lumberjacks environment to evaluate different aspects of our algorithm compared to the reference algorithms. All experiments were ran for 30.000 games (epochs), where we trained after a set amount of steps and algorithms were evaluated using the discounted reward. The experiments were not repeated multiple times due to time and computational constraints.

1. **Number of Agent Experiment**: We ran all algorithms with 2,3 and 4 agents, to understand how they handle different numbers of agents. Environment hyperparemeters included a grid size of 4, 6 trees and an observation and communication radius of 1.
2. **Environment Complexity Experiment**: Algorithms were compared using different environment complexities with grid sizes 4, 6 and 8 and 6, 10 and 14 trees, respectively. The aim

was to evaluate learning under more difficult circumstances. Environment hyperparameters included 2 agents, and an observation and communication radius of 1.

3. **Communication Experiment**: Algorithms were run with communication radii of -1 (no communication), 1, and 2 to understand how communication aids learning. Environment hyperparemeters included 2 agents, a grid size of 5, 8 trees and observation radii of 1, 1 and 2.

To validate the results in the Lumberjacks environment, we repeated the experiments in the Cat & Mouse environment, substituting the number of trees with the number of mice.

4 Results

This section presents the results of the proposed experiments proposed for the Lumberjacks and the Cat & Mouse environments. Detailed training plots for the Lumberjacks environment and a table for the maximal discounted reward averaged over 500 episodes for the Cat & Mouse environment are provided. Different representations of the results are in the Appendix A.1.

4.1 Lumberjacks

4.1.1 Number of Agents

First, we investigate capabilities of the algorithms to handle multiple agents.

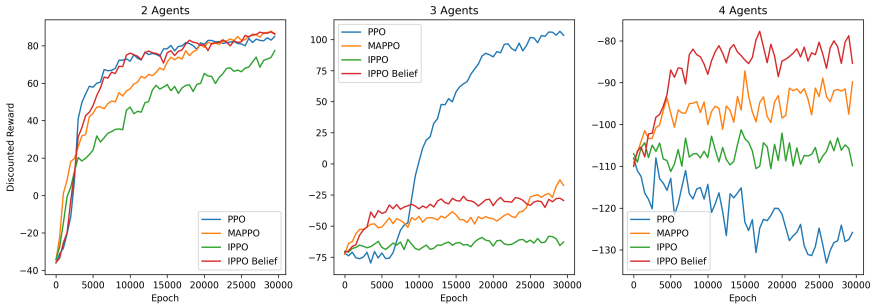

Figure 4: Number of Agents Experiment Lumberjacks

The first experiment shows that all algorithms are able to learn a good policy with two agents. Notably, our IPPO with Belief algorithm converges as quickly as fully centralized PPO with global observations and slightly faster than MAPPO. IPPO performs worst as expected.

The experiments with 3 and 4 agents show the complexity of learning a good policy with more than 2 agents. The centralized PPO algorithm learns a good policy with 3 agents but fails with 4. Our algorithm manages to achieve equal or better performance than the other reference algorithms.

4.1.2 Environment Complexity

Secondly, we study the ability of the algorithms to deal with complexity.

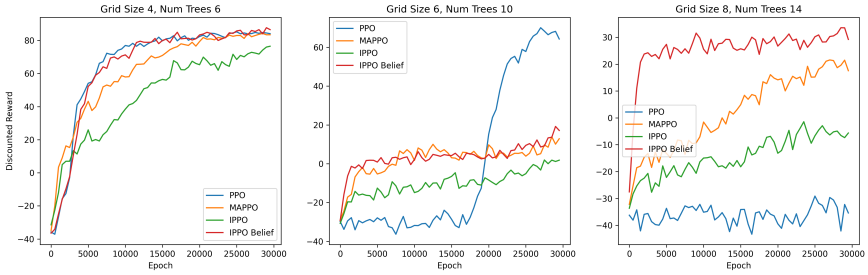

Figure 5: Environment Complexity Experiment Lumberjacks

Increasing environment complexity, such as larger grids and more trees, leads to similar behavior as adding more agents. For grid size 6 and 10 trees, PPO initially achieves low rewards before dramatically improving after 15.000 episodes. On the hardest environment PPO fails to learn a good policy in the 30.000 episodes, while our method works the best.

4.1.3 Communication

Lastly, we investigate the impact of communication on the learning process.

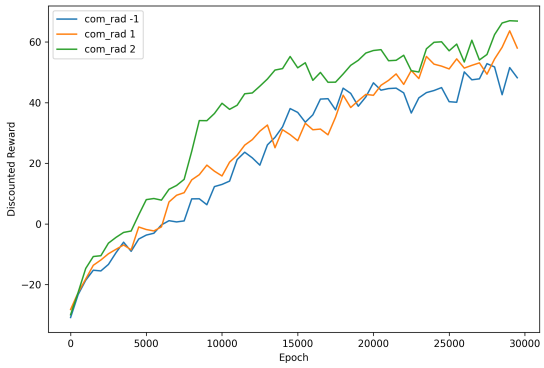

Figure 6: Communication Experiment Lumberjacks

The experiment shows that agents with a communication radius of 2 achieve the highest discounted reward after 30.000 epochs. The rewards of agents with communication radius 1 and -1 are very similar. Although there is no significant difference in maximum rewards, agents with a radius of 2 achieve higher rewards quicker.

4.2 Cat & Mouse

We repeated the same experiments in the Cat & Mouse environment that were ran in the Lumberjacks environment, substituting the number of trees with the number of mice.

4.2.1 Number of Agents

Experiments	IPPO with Belief	IPPO	PPO	MAPPO
2 Agents	130.03	126.37	110.31	**139.91**
3 Agents	219.97	206.15	158.31	**246.65**
4 Agents	302.42	287.79	208.68	**372.93**

Table 1: Number of Agents Experiment - Maximum Average Reward (500 episodes)

The experiments demonstrate that in this environment the MAPPO algorithm significantly outperforms the PPO algorithm. The IPPO with Belief algorithm consistently learns effective policies, surpassing the other two reference algorithms in all tests. Interestingly, the performance of the PPO algorithm is notably poorer than in the Lumberjacks environment.

4.2.2 Environment Complexity

Experiments	IPPO with Belief	IPPO	PPO	MAPPO
Grid Size 4, 6 Mice	126.67	124.8	99.11	**134.42**
Grid Size 6, 10 Mice	**196.91**	180.76	143.9	188.57
Grid Size 8, 14 Mice	218.09	**227.41**	138.316	210.6

Table 2: Environment Complexity Experiment - Maximum Average Reward (500 episodes)

The experiment assessing the ability of the algorithms to cope with higher complexity presents results similar to those of the Number of Agents experiment. The IPPO with Belief algorithm again achieves competitive rewards compared to the other algorithms. Additionally, the IPPO algorithm performs significantly better in this environment than in the Lumberjacks environment.

5 Discussion

The experiments conducted in the Lumberjack environment demonstrated that incorporating a belief state and communication into the IPPO algorithm enhances the learning capabilities of the agents compared to the original formulation. This modification enables IPPO to effectively compete with global state space algorithms such as PPO and MAPPO, which utilize centralized policies. We hypothesize that centralized algorithms do not significantly outperform our localized approach because the state and action spaces, which expand rapidly with the number of agents, complicate the planning process. Additionally, the minor improvements of using higher communication radii, suggest limited efficacy, possibly due to the requirement for agents to be in close proximity for effective communication.

Validation using the Cat & Mouse environment demonstrated that the IPPO with Belief algorithm also performs robustly in diverse settings. Compared to the Lumberjack environment, the IPPO algorithm exhibited superior performance, while PPO showed deteriorated results. We attribute this to the increased independence of agents in the Cat & Mouse environment, which likely influenced the algorithms' effectiveness. Moreover, extensive testing with various configurations of the PPO algorithm revealed a significant dependence on hyperparameter settings, which affected performance across different environments. These observations are further detailed in the Appendix A.4.

6 Future Work

In our work, we have focussed on communicating the beliefs about the current state of the environment, but our framework is not restricted to this type of information alone. Sharing various types of data, such as the parameters of actor and critic networks, could enhance cooperation further by allowing agents to exchange detailed insights about their experiences within the environment.

Another promising research direction involves establishing the theoretical properties of the newly introduced Multi-Agent MDP with Local Observation & Communication. We hypothesize that with informative local observations and frequent communication, it is possible to achieve convergence. This aspect warrants a thorough theoretical investigation to better understand the dynamics and potential guarantees of the proposed approach.

References

[1] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11), 2021. ISSN 2076-3417. doi: 10.3390/app11114948. URL https://www.mdpi.com/2076-3417/11/11/4948.

[2] Eugene Vinitsky2 Jiaxuan Gao Yu Wang1 Alexandre Bayen Yi Wu Chao Yu1, Akash Velu. The surprising effectiveness of ppo in cooperative multi-agent games. 2021. URL https://arxiv.org/pdf/1706.02275.

[3] Denys Makoviichuk Viktor Makoviychuk Philip H.S. Torr Mingfei Sun Shimon Whiteson Christian Schroeder de Witt, Tarun Gupta. Is independent learning all you need in the starcraft multi-agent challenge? 2020. URL https://arxiv.org/abs/2011.09533.

[4] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms. 2017. URL https://arxiv.org/abs/1707.06347.

[5] Anurag Koul. ma-gym: Collection of multi-agent environments based on openai gym. https://github.com/koulanurag/ma-gym, 2019.

[6] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020.

[7] Christian Schroeder de Witt Gregory Farquhar Jakob Foerster Shimon Whiteson Tabish Rashid, Mikayel Samvelyan. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. 2018. URL https://arxiv.org/pdf/1803.11485.

[8] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL https://zenodo.org/record/8127025.

[9] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5872–5881. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/zhang18n.html.

Contributions

Owen Du developed the Cat & Mouse environment and adapted the PPO algorithm to run more efficiently on the discrete environment.

Leander Diaz-Bone conceptualized the notion locality used in this project, developed the IPPO with Belief algorithm and worked on the implementation of the algorithm and the experiments.

Georg Ye conducted research on available multi-agent RL algorithms, implemented and tested them, and performed hyperparameter tuning.

A Appendix

A.1 Further Results

In the following section we provide the results already presented in the Results section in the respectively different format.

A.1.1 Lumberjacks

Experiments	IPPO with Belief	IPPO	PPO	MAPPO
2 Agents	87.41	77.39	85.01	**87.71**
3 Agents	-26.04	-58.18	**106.57**	-12.82
4 Agents	**-77.72**	-101.28	-107.984	-87.18

Table 3: Number of Agents Experiment Lumberjacks - Maximum Average Reward (500 episodes)

Experiments	IPPO with Belief	IPPO	PPO	MAPPO
Grid Size 4, 6 Trees	**87.68**	76.5	85.20	84.21
Grid Size 6, 10 Trees	19.21	1.92	**70.08**	13.83
Grid Size 8, 14 Trees	**33.56**	-1.44	-29.12	21.56

Table 4: Environment Complexity Experiment Lumberjack - Maximum Average Reward (500 episodes)

A.1.2 Cat & Mouse

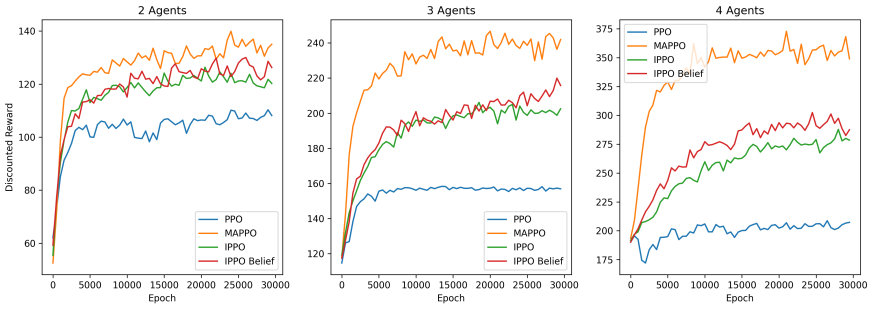

Figure 7: Number of Agents Experiment Cat & Mouse

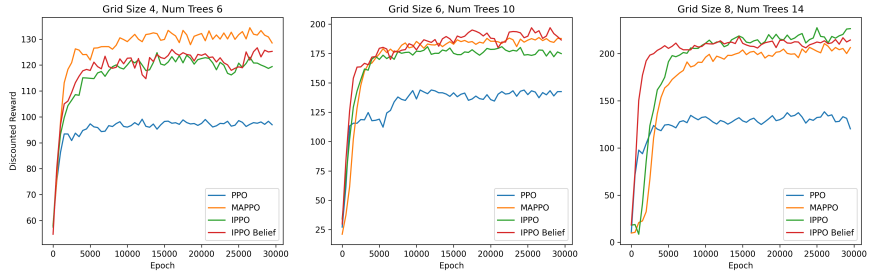

Figure 8: Environment Complexity Experiment Cat & Mouse

A.2 Implementation Details

In this part we provide implementation details of environments and algorithms that were important to the success of the learning.

A.2.1 Belief State Distribution Implementation

The implementation of the belief state distribution is a critical component of the proposed IPPO with Belief algorithm, as it determines how the agents approximate the global state. The implementation for the Lumberjacks and the Cat & Mouse environment are notably similar, due to the grid structure common to both environments. We will outline the shared aspects and highlight the differences below.

Each agent maintains its own local belief distribution over the global state. For the Lumberjacks environment, this includes a categorical distribution for the position of each agent across each grid position. Additionally, a categorical distribution for each tree strength, assigning probabilities to each grid position that indicate the likelihood of containing a tree of a specific strength. Each new local observation is used to update the probabilities associated with the positions of agents and trees. For trees, this process involves assigning a probability of zero to observed empty positions and one to positions containing a tree. For agents, when an agent is not observed, we infer the probability of their current position from the previous distribution, using a random walk to compute these probabilities. During the communication phase, the belief states from various agents are merged by multiplying the positional distributions to derive a unified distribution that can be utilized by all communicating agents.

The implementation of the belief state distribution for the Cat & Mouse environment is quite similar to that described above. A significant difference is the need to update the position distributions for the moving mice, analogous to the updates for agent positions. Additionally, before finalizing the belief state, we aggregate all positional distributions for the mice to simplify the belief state while retaining essential information. The algorithm benefits significantly from centralizing all position distributions relative to the agent's position.

In a continuous environment, such as the continuous Cat & Mouse environment, modeling the positions as normal distributions is a practical approach. If an actor is not observed, the variance of the distribution is increased to reflect the uncertainty. To facilitate communication of these position distributions, we identify the normal distribution that best approximates the product of all individual position distributions.

A.2.2 Continuous Cat & Mouse Environment

In the continuous environment, we keep track of the agents and prey using lists with their position and a "caught"-flag. In local observations, we additionally use flags that track whether the respective agent or prey is within the observation radius or not. Each agent provides one float between 0 and 1, which gets translated to the direction of travel for each step. On top of catch rewards and step penalties, the agents are penalized based on the distance to the closest prey to incentivize moving in the direction of prey. In both discrete and continuous environments, we can also set a communication radius parameter, which returns for each agent a list of agents whithin communication radius. A visualization of the continuous environment can be seen below.

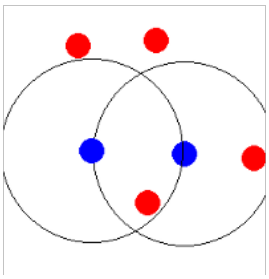

Figure 9: Continuous Cat & Mouse Environment Visualization. The blue circles represent the agents (or cats) and red circles represent the prey (or mice). The black circle in the continuous environment denotes the observation radius of the agents.

A.3 Notes on MADDPG & QMIX

MADDPG (Multi-Agent Deep Deterministic Policy Gradient) is an algorithm designed for multi-agent environments where each agent learns a policy within a shared environment (Lowe et al., 2020) [6]. MADDPG utilizes the actor-critic framework, wherein each agent has an actor network that determines actions and a critic network that evaluates the action-value function. As a centralized training with decentralized execution (CTDE) algorithm, MADDPG leverages the policies of all agents through its critic networks to provide more informed value estimates, thus allowing agents to better coordinate their actions.

Despite the promising results reported in the original MADDPG paper, our attempts to reproduce these results were unsuccessful. We tested the algorithm using both our implementation and the authors' cloned repository on the "Simple Spread (Cooperative Navigation)" environment but were unable to achieve the claimed performance. This discrepancy suggests potential issues with reproducibility or differences in experimental setups.

QMIX is another algorithm designed for multi-agent environments, specifically aimed at improving the scalability and efficiency of value-based methods (Rashid et al., 2018) [7]. QMIX builds on the Value Decomposition Networks (VDN) approach by learning a more flexible mixing function to combine individual agents' value functions into a global value function. QMIX ensures that the global Q-function is a monotonic combination of the individual Q-functions, allowing for efficient learning while maintaining the ability to act independently during execution.

We also implemented QMIX and tested it but did not manage to get the agents to learn effectively. After extensive debugging, we decided to focus on the PPO family for our benchmarks due to time constraints and concerns about encountering similar reproducibility issues as with MADDPG. The PPO family, known for its proven stability and effectiveness, provided a more reliable basis for our experiments.

A.4 Notes on PPO in the Simple Spread and Continuous Cat & Mouse Environment

In our implementation, we initially tested PPO with a single hidden layer, which did not yield effective learning. Upon increasing the number of hidden layers to two, the agents began to learn effectively. We experimented with different activation functions and found that the Rectified Linear Unit (ReLU), as suggested in the original PPO paper, did not perform well. Instead, using the Tanh activation function resulted in successful learning.

We also tested various hidden layer sizes, specifically 128 and 256 units. Both configurations yielded similar results, but the 256-unit configuration took longer to converge. Therefore, we chose a hidden layer size of 128 for our experiments to balance performance and convergence time.

Early in the project, we utilized the Simple Spread (Cooperative Navigation) environment, which was used in the MADDPG paper. This environment features N agents and N landmarks, where each agent must navigate to a landmark without colliding with others. Surprisingly, among the various algorithms we tested, only MAPPO managed to solve this environment. Even with the number of agents N set to 2, PPO failed to achieve the desired outcomes. This unexpected behavior prompted us to discontinue using this environment.

Our focus then shifted to using the Lumberjacks environment and our custom Cat & Mouse environment for benchmarking. This decision was driven by the inconsistency and unpredictability observed in the Simple Spread environment.

PPO can be adapted from discrete to continuous settings by removing the softmax layer, thereby returning the mean of the action distribution. In our experiments with the Cat & Mouse setting, we noticed that the agent's learning performance significantly improved in the discrete setting. Consequently, we decided to concentrate our efforts on the discrete setting to optimize learning outcomes.

A.5 Code

All implementations of algorithms, environments and experiments can be found in the following GitHub. Please refer to the README file for details on location of the implementations.

https://github.com/LeanderDiazBone/decentralized_marl