Asynchronous Multi-Agent Reinforcement Learning for Efficient Real-Time Multi-Robot Cooperative Exploration

Chao Yu
Tsinghua University
Beijing, China
zoeyuchao@gmail.com

Xinyi Yang
Tsinghua University
Beijing, China
yang-xy20@mails.tsinghua.edu.cn

Jiaxuan Gao
Shanghai Qi Zhi Institute
Tsinghua University, China
samjia2000@gmail.com

Jiayu Chen
Tsinghua University
Beijing, China

Yunfei Li
Tsinghua University
Beijing, China

Jijia Liu
Tongji University
Shanghai, China

Yunfei Xiang
Tsinghua University
Beijing, China

Ruixin Huang
Tsinghua University
Beijing, China

Huazhong Yang
Tsinghua University
Beijing, China

Yi Wu
Tsinghua University
Shanghai Qi Zhi Institute, China
jxwuyi@gmail.com

Yu Wang
Tsinghua University
Beijing, China
yu-wang@tsinghua.edu.cn

ABSTRACT

We consider the problem of cooperative exploration where multiple robots need to cooperatively explore an unknown region as fast as possible. Multi-agent reinforcement learning (MARL) has recently become a trending paradigm for solving this challenge. However, existing MARL-based methods adopt *action-making steps* as the metric for exploration efficiency by assuming all the agents are acting in a fully synchronous manner: i.e., *every* single agent produces an action *simultaneously* and *every* single action is executed *instantaneously* at each time step. Despite its mathematical simplicity, such a synchronous MARL formulation can be problematic for real-world robotic applications. It can be typical that different robots may take slightly different wall-clock times to accomplish an atomic action or even periodically get lost due to hardware issues. Simply waiting for every robot being ready for the next action can be particularly *time-inefficient*. Therefore, we propose an asynchronous MARL solution, *Asynchronous Coordination Explorer (ACE)*, to tackle this real-world challenge. We first extend a classical MARL algorithm, multi-agent PPO (MAPPO), to the asynchronous setting and additionally apply action-delay randomization to enforce the learned policy to generalize better to varying action delays in the real world. Moreover, each navigation agent is represented as a team-size-invariant CNN-based policy, which greatly benefits real-robot deployment by handling possible robot lost and allows bandwidth-efficient intra-agent communication through low-dimensional CNN features. We first validate our approach in a grid-based scenario. Both simulation and real-robot results show that ACE reduces over 10% actual exploration time compared with classical approaches. We also apply our framework to a high-fidelity visual-based environment, Habitat, achieving 28% improvement in exploration efficiency.

KEYWORDS

Multi-Agent Reinforcement Learning; Asynchronous Decision Making; Cooperative Exploration

ACM Reference Format:
Chao Yu, Xinyi Yang, Jiaxuan Gao, Jiayu Chen, Yunfei Li, Jijia Liu, Yunfei Xiang, Ruixin Huang, Huazhong Yang, Yi Wu, and Yu Wang. 2023. Asynchronous Multi-Agent Reinforcement Learning for Efficient Real-Time Multi-Robot Cooperative Exploration. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Exploration is a fundamental task for building intelligent robot systems, which has been applied in many application domains, including rescue [12], autonomous driving [1], drone [27], and mobile robots [18]. In this paper, we consider a multi-robot cooperative exploration task, where multiple homogeneous robots simultaneously explore an unknown spatial region in a cooperative fashion. Learning the optimal cooperative strategies can be challenging due to the existence of multiple robots. These robots must effectively distribute the exploration workload so that they can always navigate towards different spatial regions to avoid trajectory conflicts, which accordingly leads to a remarkably higher exploration efficiency than the single-robot setting.

Multi-agent reinforcement learning (MARL) has been a trending approach to tackle this cooperative exploration challenge. RL-based methods directly learn neural policies end to end by interacting with a simulated environment for policy improvement. Compared with planning-based solutions [2, 19, 24] which require non-trivial implementation heuristics and expensive inference computation at

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

execution time, RL-based methods [5, 19] provide strong representation capabilities of complex strategies and negligible inference overhead once the policies are trained.

Classical multi-agent RL algorithms typically adopt a synchronous algorithmic framework, i.e., all the agents are making actions at the same time, and all the actions will be executed immediately at each time step, leading to the next action-making step for future actions. This process is mathematically formulated as a decentralized Markov decision process, which is widely adopted in multi-agent RL literature. Although such a mathematical framework is simple and elegant, it can be problematic for real-world multi-robot exploration tasks. For real robot systems, each actual action is never atomic and may take varying times to finish. These action delays can be more severe due to unexpected network communication traffic or hardware failure. Simply following the synchronous setting, i.e., waiting until every robot is ready before making new actions, can be particularly real-time inefficient. Therefore, an ideal RL framework for real-world use should be asynchronous, i.e., whenever an agent finishes action execution, it should immediately generate the next action, and the learned strategy should effectively enable such an asynchronous action-making process.

In this paper, we propose a novel asynchronous multi-agent RL-based solution, *Asynchronous Coordination Explorer (ACE)*, to tackle the real-world multi-robot exploration task. We first extend a classical MARL algorithm, multi-agent PPO (MAPPO), to the asynchronous setting to effectively train the multi-agent exploration policy, and additionally leverage an action-delay randomization technique to enable better simulation to real-world generalization. Moreover, we design a communication-efficient Multi-tower-CNN-based Policy (MCP) for each agent. In MCP, a CNN module is applied to each agent's local information to extract features, and a fusion module combines each agent's features to produce an action. During execution time, efficient intra-agent communication can be achieved via directly exchanging low-dimensional features extracted by the weight-sharing CNN module. Another benefit of MCP is to tackle varying team sizes, which may occur when agents go offline in real-world applications.

We conduct experiments in a grid-based multi-room scenario both in simulation and our real-world multi-robot laboratory, where strategies learned by ACE significantly outperform both classical planning-based methods and neural policies trained by synchronous RL methods. In particular, *ACE reduces* 10.07% ***real-world*** *exploration time* than the synchronous RL baseline and *reduces* 33.86% ***real-world*** *exploration time* than the fastest planning-based method with 2 Mecanum steering robots. Besides, we extend ACE to a vision-based environment, Habitat, verifying the effectiveness of the asynchronous training mechanism when applied to more complicated environments. More demonstrations can be seen on our website: https://sites.google.com/view/ace-aamas.

2 RELATED WORK

2.1 Cooperative Exploration

Multi-agent cooperative exploration is an important task for building intelligent mobile robot systems. There are a large number of works developing planning-based methods for this problem [8, 24, 34], but they typically rely on manually designed heuristics [2, 7, 31] and are limited in expressiveness to learn more complex cooperation strategies. Another popular line of research is deep MARL-based methods which leverage the expressiveness power of neural networks to learn non-trivial cooperative exploration skills [13, 28–30, 35, 41]. Note that most existing MARL-based methods assume synchronous action execution among all the agents or consider atomic actions, which we believe is due to the synchronous design of most simulated RL environments. However, such synchronous design does not reflect the real-world multi-agent systems, where agents take actions at different real times due to network delay and unexpected hardware take-downs. We propose an *asynchronous* MARL exploration framework in this work to better match real-world applications. [15] considers an asynchronous decision-making mechanism for large-scale problems and proposes an improved Monte Carlo Search method to solve this problem. A concurrent work formulates a set of asynchronous multi-agent actor-critic methods that allow agents to directly optimize asynchronous policies [33], while we design an asynchronous MARL training framework combined with action-delay randomization. We also notice a recent trend of developing asynchronous simulation [10, 36], which we hope can further accelerate the advances in applying MARL methods to the real world.

2.2 Sim2real Transfer

It is often challenging to directly deploy policies trained in simulation to the real world since there is always a mismatch between simulation and reality. Domain randomization is a simple but effective technique to fill the reality gap, which creates a variety of simulated environments with randomized properties such as physical dynamics [17, 21] and visual appearances [16, 22], and tries to train RL that can perform well among all of them. We also adopt the idea of domain randomization, and randomly delay the execution of each agent in simulation to model the uncertain delay between policy computation and actual action execution in the real world. The action delay technique is also adopted in other domains such as model-based RL [4] and reactive RL [23].

3 PRELIMINARY

3.1 Task Setup

We study the task of real-time multi-robot cooperative exploration, where a team of robots aims to explore an unknown environment exhaustively as fast as possible. The real-time multi-robot task is with an asynchronous nature, i.e., different robots do not take actions and receive the next-step observations at the same time. A real robot often requires non-fixed time to execute an action, and unexpected hardware failure can cause random delays. Besides, under the standard bi-level control setting in robot navigation [11, 24, 34, 40], each action is a goal position to reach and typically requires varying number of atomic steps to accomplish, thus exacerbating the asynchronous issue. The asynchronous execution is not considered by classical multi-agent reinforcement learning (MARL) works. It is typically assumed all agents take actions at the same *action-making step* and do not take the action execution time into account. In the traditional MARL literature [38], the task is usually formulated as a decentralized partially observable Markov decision process (Dec-POMDP), which is unable to capture the asynchronous property in our setting.

3.2 Problem Formulation

We model the asynchronous multi-agent cooperative exploration task as a decentralized partially observable Semi-Markov decision process (Dec-POSMDP) [15] with shared rewards. We adopt a modular action execution scheme [3, 14] which consists of bi-level actions for robust deployment in real-world robot systems. A macro action (MA), i.e., global goal, is generated in the action-making step. Several atomic actions, i.e., execution actions, are followed to perform under the guidance of the MA.

To avoid notation ambiguity, we use $p^{(i)}$ to denote a parameter p related to the i-th agent, and $\bar{p} = (p^{(1)}, p^{(2)}, \cdots, p^{(n)})$ to denote joint parameters for multiple agents thereafter. A Dec-POSMDP is defined by a set of elements $\langle D, \bar{U}, \bar{B}, P, \bar{R}^\tau \rangle$. $D = \langle \bar{S}, \bar{A}, \bar{\Omega}, \bar{O}, \bar{R}, P, n, \gamma \rangle$ defines the decentralized partially observable Markov decision processes (Dec-POMDP), where $\bar{S}$ is the joint state space, $\bar{A}$ is joint atomic action space, $\bar{\Omega}$ is the observation space, $O^{(i)}(o^{(i)}|s, a^{(i)})$ denotes the observation probability function for agent i, $\bar{R} : \bar{S} \times \bar{A} \to \mathbb{R}$ is the joint reward function, n is the number of agents, P is the state transition probability. $\bar{U}$ is joint macro-action space. A macro action $u^{(i)}$ is a high-level policy that can generate a sequence of atomic actions $a_t \sim u^{(i)}(H_t^{(i)})$ for any t when $u^{(i)}$ is activated, where $H_t^{(i)}$ is the individual action-observation history till t. $\bar{B}$ denotes the stop condition of MA and $B^{(i)}(u^{(i)})$ is represented as a set of action-observation histories of an agent i. If $H_t^{(i)} \in B^{(i)}(u_t^{(i)})$ holds, $u_t^{(i)}$ terminates and the agent generates a new MA. $\bar{R}^\tau$ is the macro joint reward function: $\bar{R}^\tau(\bar{s}, \bar{u}) = \mathbb{E}\left[\sum_{t=0}^{\bar{\tau}_{end}} \gamma^t \bar{R}(\bar{s}_t, \bar{a}_t) | \bar{a}_t \sim \bar{u}(\bar{H}_t)\right]$ where $\bar{\tau}_{end} = \min_t\{t : H_t^{(i)} \in B^{(i)}(u^{(i)})\}$.

The solution of a Dec-POSMDP is a joint high-level decentralized policy $\bar{\phi} = (\phi^{(1)}, \cdots, \phi^{(n)})$ where each $\phi^{(i)}$ produces an MA $\phi^{(i)}(H_t^{(i)}) \in U^{(i)}$ given individual action-observation history $H_t^{(i)}$. In the beginning of an episode, an initial MA is computed as: $u_{t_0}^{(i)} = \phi^{(i)}(H_{t_0}^{(i)})$. At action-making step $k > 0$, the agent generates a new MA $u_{t_k}^{(i)} = \phi^{(i)}(H_{t_k}^{(i)})$ if the stop condition is met, i.e. $H_{t_k}^{(i)} \in B^{(i)}(u_{t_{k-1}}^{(i)})$. Otherwise, the agent continues to use the previous MA: $u_{t_k}^{(i)} = u_{t_{k-1}}^{(i)}$. In the time range $[t_k, t_{k+1})$, the agent interacts with the environment with atomic actions sampled from MA: $a_t^{(i)} \sim u^{(i)}(H_t^{(i)})$. Finally, the goal of Dec-POSMDP is to maximize the accumulative discounted reward: $\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{t_k} \bar{R}^\tau(\bar{s}_{t_k}, \bar{u}_{t_k}) | \bar{\phi}, \bar{s}_0\right]$ where $t_0 = 0$ and $t_k = \min_t\{t > t_{k-1} : H_t^{(i)} \in B^{(i)}(u_{t_{k-1}}^{(i)})\}$ for $k \geq 1$. A more detailed definition can be found in [15].

In our *asynchronous* setting, t is the real time, not the discrete time step as in common *synchronous* RL. Our setting is more time-efficient and robust to hardware faults. Take a 2-agent case as an example (see Fig. 1), in the synchronous setting, the agents can only transmit data (blue and green arrows) and perform policy inference (orange arrow) after both of them have finished the previous action execution. The system execution speed is bottle-necked by the agent with the longest execution time. Worse still, the whole system will get stuck if one agent goes offline unexpectedly. By contrast, agents take actions in a distributed manner in an asynchronous setting. Each agent can request data from other agents and conduct policy inference immediately after it finishes its own action execution. This asynchronous setting is more time-efficient for multi-agent exploration tasks, and will not be blocked by dynamic changes such as agents going offline.

3.3 Connection to Conventional MARL

In the conventional MARL literature [38], the problem formulation is typically under decentralized partially observable Markov decision process (Dec-POMDP), which assumes synchronized actions. In this work, we also focus on the multi-agent setting and assume a shared reward function and dynamic transitions. However, different from synchronous MARL which assumes all agents execute actions simultaneously, we consider the asynchronous nature in the practical multi-robot scenarios.

We will adapt a popular MARL algorithm, Multi-Agent Proximal Policy Optimization (MAPPO) [38], from the conventional setting to our asynchronous setting. Conventional MAPPO follows the Centralized Training and Decentralized Execution (CTDE) paradigm, in which agents make decisions with individual observations and update the joint policy with global information in a centralized manner. Under the framework of Dec-POMDP, MAPPO requires all agents taking actions synchronously at each discrete time step, and the state transits according to actions from all agents: $s_t \sim P(\cdot|s_{t-1}, \bar{a}_{t-1})$. It aims to find a joint policy $\bar{\pi}$ that maximizes the accumulated discounted reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \bar{R}(s_t, \bar{a}_t) | a_t^{(i)} \sim \pi^{(i)}(H_t^{(i)})\right]$. Different from MAPPO, Async-MAPPO is designed for the asynchronous setting, where there are no centralized environment steps.

4 METHODOLOGY

To better model the asynchronous nature of real-world multi-agent exploration problems, we present Asynchronous Coordination Explorer (ACE). ACE consists of 3 major components: (1) Async-MAPPO for MARL training, (2) action-delay randomization for zero-shot generalization in the real world, and (3) multi-tower-CNN-based policy representation for efficient communication.

4.1 Async-MAPPO

We extend an on-policy MARL algorithm MAPPO [37] to our asynchronous setting, which we call Async-MAPPO. The pseudo-code of Async-MAPPO is shown in Algo. 1. Compared with the setting of MAPPO, both policy execution and data collection are not necessarily time-aligned among different agents, and we implement the asynchronous action-making and replay buffer as follows.

- We design a bi-level execution scheme. In ACE, agents perform atomic actions under the guidance of global goals (macro actions). Instead of receiving the reward, local observation, and states immediately after executing an atomic action, Async-MAPPO accumulates the reward between action-making steps and only takes observation and states at each macro action.
- We implement asynchronous buffer insertion, in contrast to the synchronous scheme in original MAPPO as shown in Fig. 1. The original MAPPO assumes synchronous execution of all the agents; in each time step, all the agents take actions simultaneously, and the trainer waits for all the new transitions before inserting them into a centralized data buffer for RL training. In Async-MAPPO, different agents may not take actions at the same time (some agents may even get stuck and cannot return new observations at all), which makes it infeasible for the

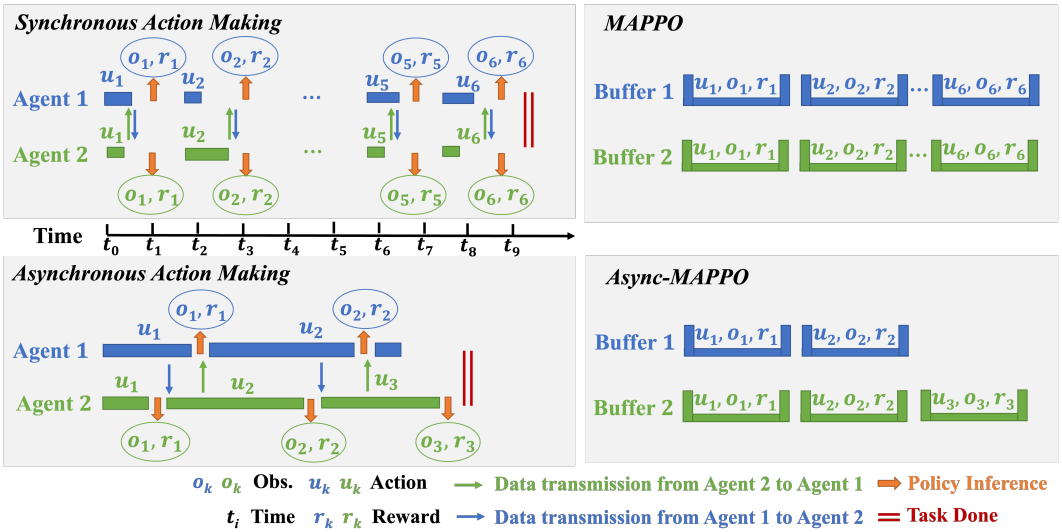

Figure 1: Comparison of *asynchronous* and *synchronous* action making.

Algorithm 1: Async-MAPPO

Initialize the policy π;
while $step \leq step_{max}$ **do**
  set data buffer $D = \{\}$;
  for $i = 1$ *to batch_size* **do**
    Reset the environment;
    Create N empty caches $C = [[], \ldots, []]$;
    for $t = 1$ *to* T **do**
      for *all agents* $i = 1$ *to* N **do**
        if *agent* i *replans macro action* **then**
          $b \leftarrow$ agent i's b-th macro actions;
          $s_b^{(i)} \leftarrow State, o_b^{(i)} \leftarrow Observation$;
          $C_i += [s_{b-1}^{(i)}, o_{b-1}^{(i)}, u_{b-1}^{(i)}, \hat{R}_b^{\tau(i)}, s_b^{(i)}, o_b^{(i)}]$;
          $p_b^{(i)} = \pi(o_b^{(i)})$;
          Update macro action $u_b^{(i)} \sim p_b^{(i)}$;
        Execute atomic action $a_t^{(i)} \sim u_b^{(i)}$;
    Compute reward-to-go and insert data into D;
  Update π on MAPPO loss;

trainer to collect transitions in the original synchronous manner. Therefore, we allow each agent to store its own transition data in a separate cache and periodically push the cached data to the centralized data buffer. We can then run the standard MAPPO training algorithm over this buffer.

4.2 Action-Delay Randomization

When training in traditional simulators, agents can always take execution steps synchronously without considering different action execution costs. Moreover, real-world action delays such as hardware failure and network blocking are not simulated. These problems cause a large gap for deploying trained agents from simulation to reality. To reduce this gap, we apply action-delay randomization during simulation. In the end of each action-making step, we force each agent to wait for a random period from 3 to 5 execution steps in grid-based environments, and from 10 to 15 execution steps in Habitat before querying the next macro action.

4.3 Multi-Tower-CNN-Based Policy

The Multi-tower-CNN-based Policy (MCP) is utilized to generate macro actions, i.e., global goals in ACE. As illustrated in Fig. 3, MCP consists of 3 parts, i.e., a CNN-based local feature extractor, an attention-based relation encoder, and an action decoder.

The local feature extractor is a weight-sharing 3-layer CNN and can extract a $G \times G \times 4$ feature embedding from each agent's $S \times S \times 7$ local information, which includes one obstacle channel, one explored region channel, one-hot location channel, one trajectory channel to represent the history trace, and three agent-view channels of the agent's local observation.

The agents transmit extracted feature embedding instead of the raw local information, which greatly reduces communication traffic by $1 - \frac{G \times G \times 4}{S \times S \times 7} = 1 - \frac{4}{7\alpha^2}$ times where $\alpha = S/G$. For example, we adopt $G = 5$ in grid-based environments, thus the communication traffic reduces $\sim 97\%$ in $S = 25$ maps and $\sim 93\%$ in $S = 15$ maps.

The relation encoder aims to aggregate the extracted feature maps from different agents to better capture the intra-agent interactions. In team-based exploration, an agent should not only spot undiscovered areas but also inter-teammates' movement for better scheduling among agents. We adopt a simplified Transformer [26] block as the team-size-invariant relation encoder. Inspired by the vision transformer model [9], we apply multi-head cross-attention [25] to derive a single team-size-invariant representation of size $G \times G \times 4$, as shown in Fig. 3.

Finally, the action decoder predicts the agent's policy from the aggregated representation as a multi-variable Categorical distribution to select a grid cell g from a plane as the global goal (u_x, u_y). Note that in Habitat, in order to produce accurate global goals, we adopt a spatial action space with three separate action heads, i.e., two discrete region heads for choosing a grid cell g, which are the same as grid-based environments, and two additional continuous point heads for outputting a coordinate (Δ_x, Δ_y), indicating the

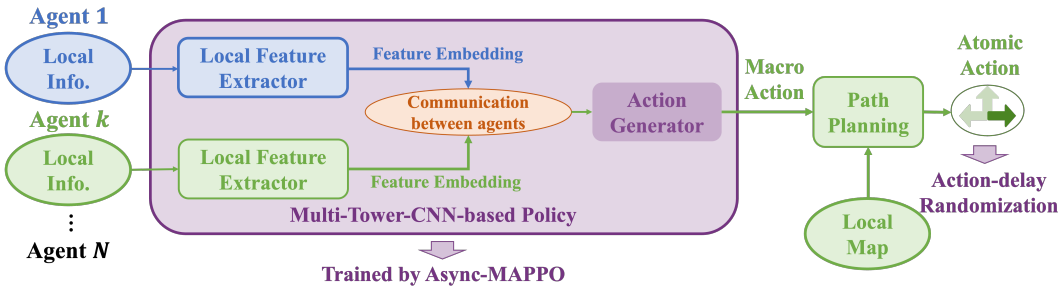

Figure 2: Overview of Asynchronous Coordination Explorer (ACE).

relative position of the global goal within the selected region g. Details of MCP in Habitat can be found in Appendix A.2.

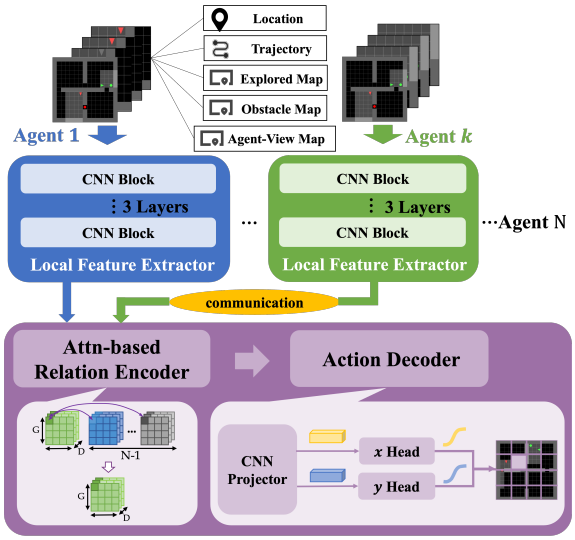

Figure 3: Workflow of *Multi-tower-CNN-based Policy* (MCP), including a CNN-based local feature extractor, a relation encoder, and an action decoder.

4.4 Overall Architecture

As shown in Fig. 2, each agent observes the local information and requests the latest feature embedding from other agents, which is output by the weight-sharing local feature extractor, at each action-making step. That is, agents only need to transfer the low-dimensional feature embedding, instead of the entire local information. The multi-tower-CNN-based policy, which is trained by Async-MAPPO, generates the next macro action, i.e., global goal, at each action-making step, and the agent performs path planning on the local map according to the global goal, outputting the atomic action at each time step. Note that agents could go offline in multi-agent tasks due to unexpected network communication traffic or hardware failure.

5 ENVIRONMENT DETAILS

5.1 Environment Setting

Grid-based scenario: As shown in Fig. 4, we implement a multi-agent exploration task based on the GridWorld simulator [6], which was originally designed for synchronous settings. We consider two different map sizes, which are 15×15 with $4 \sim 9$ random rooms and 25×25 with $4 \sim 25$ random rooms. All the agents are uniform randomly spread over the map in the beginning. The local information of each robot is fed to the RL-trained policy or planning-based methods to generate a global goal and $A^\star$ algorithm is utilized to plan 5 atomic actions on the local map to follow the global goal.

We also set up a 15×15 real-world grid map which is the same as the grid-based simulation, and each grid is 0.31m long, as shown in Fig. 4. Our robots are equipped with Mecanum steering and an NVIDIA Jetson Nano processor. The locations and poses of robots are tracked by OptiTrack cameras and the Motive motion capture software. After training a policy in the grid-based simulator under 15×15 map with random rooms, we directly deploy it to the real-world robot system. Each real robot executes in a distributed and asynchronous manner. The robot adopts a request-send mechanism to obtain the newest feature embedding of other agents through ROS topic upon finishing all atomic actions.

Habitat: We adopt map data from the Gibson dataset [32] while the visual signals and dynamics are simulated by Habitat [20]. We follow the same environment configuration in [39] and use a pre-trained neural SLAM model to predict the robot pose and the local map. Full details of Habitat can be found in Appendix A.

5.2 Observation Space

The input of RL-trained MCP is an $S \times S$ image with 7 channels, where S is the max size of the map. The channels represent obstacles, the explored mask, the agent location, the trajectory, and three $H \times W$ agent-view. Note that each agent only maintains its locally observed information, which is memory and communication-efficient for real-world deployment.

5.3 Action Space

The overall exploration framework is hierarchical, with a global goal (macro action) followed by several atomic actions towards the goal. The action of the policy is to generate a global goal (u_x, u_y) chosen in the map, representing a discrete grid in grid-based environments or a continuous location in Habitat [20]. The available atomic actions are moving forward, turning left, and turning right provided by the simulator.

5.4 Reward Function

The team-based reward function is the sum of the coverage reward, success reward, and overlap penalty. Let $Ratio^t$ denote the total coverage ratio at time t, Exp_a^t be the explored map by agent a and

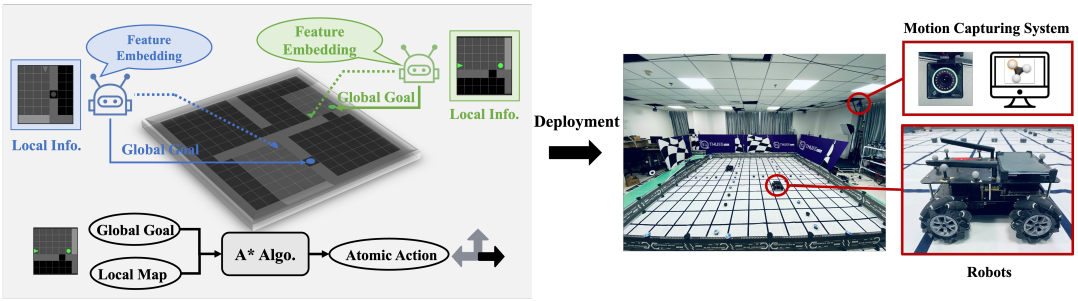

Figure 4: The illustration of the grid-based simulator and real-world robot system.

Exp^t denote the merged explored map by all agents. Both Exp^t and Exp_a^t are sets of explored areas. The reward terms are defined as follows.

- **Coverage Reward:** It is proportional to the size of the newly discovered region by the team $Exp^t \backslash Exp^{t-1}$.
- **Success Reward:** Agent a gets a success reward of $Ratio^t$ when $C\%$ coverage ratio is reached, which $C = 98$ in the grid-like simulator and $C = 90$ in Habitat[1].
- **Overlap Penalty:** The overlap penalty $r_{overlap}$ is designed to penalize repetitive exploration and encourage cooperation with others. It is defined as

$$r_{overlap} = \begin{cases} -A_{overlap} \times 0.01, & Ratio^t < 0.9 \\ 0, & Ratio^t \geq 0.9 \end{cases},$$

where $A_{overlap}$ is the increment of the overlapped explored area between agent a and other agents. The overlapped area between agent a and agent w is $Overlap_{a,w}^t = Exp_a^t \cap Exp_w^t$, and $A_{overlap} = \sum_{w \in \{1,\cdots,n\} \backslash \{a\}} Overlap_{a,w}^t \backslash Overlap_{a,w}^{t-1}$.

6 EXPERIMENT RESULTS

6.1 Training Details

In the simulation, every RL policy is trained with $50M$ steps in the grid-based simulator and $100M$ steps in Habitat over 3 random seeds. All results are averaged over a total of 300 testing episodes (100 episodes per random seed). As for real-world testing, we randomly generate 10 maps of size 15×15 and test 5 times for each map. In synchronous action-making cases, agents perform action-making at the same time and wait for all other agents to finish. In asynchronous action-making cases, agents do not wait for others and perform both macro and atomic actions independently.

6.2 Evaluation Metrics

The most important metric in our experiment is *Time*, which is the running time for the agents to reach a $C\%$ coverage ratio. We report wall-clock time in the real world, and report an estimated statistical running time in simulation: turning left or right takes $0.5s$; stepping forward takes $1s$. Policy inference time is fixed to $0.1s$ for both RL and planning-based methods thus the results can better reflect the difference between asynchronous and synchronous settings.

We also consider 3 additional statistics metrics to capture different characteristics of a particular exploration strategy. These metrics are only for analysis, and we primarily focus on *Time* as our performance criterion.

- *Accumulative Coverage Score* (ACS): The overall exploration progress throughout an episode computed as $A_T = \int_{t=0}^{T} Ratio^t$, where T is the max running time. Higher *ACS* implies faster exploration.
- *Coverage*: the *final* ratio of explored area when an episode terminates. Higher implies more exhaustive exploration.
- *Overlap*: the ratio of the overlapped region explored by *multiple* agents to the current explored area when C% coverage is reached. Lower *Overlap* implies better credit assignment.

All metrics are calculated with the running time t, i.e., the estimated statistical time in simulation and wall-clock time in the real world. Each score is reported as "mean (standard deviation)".

6.3 Baselines

We consider 4 popular planning-based competitors, including a utility-maximizing method (*Utility*) [11], a search-based nearest-frontier method (*Nearest*) [34], a rapid-exploring-random-tree-based method (*RRT*) [24], and an artificial potential field method (*APF*) [40] which applies resistance forces among agents as a cooperation mechanism. Note that APF is a multi-agent baseline while the other three are commonly used for single-agent tasks. Moreover, all baselines use global information to do planning after every macro action. Different from ACE, they are not learning-based and are all designed for asynchronous execution.

6.4 Grid-Based Scenario

6.4.1 ***Main Results.*** Experiment results with 2 agents in the grid-based simulator under synchronous and asynchronous training are provided in Table 1. In both settings, ACE outperforms planning-based baselines with $\geq 10\%$ less *Time*, full *Coverage*, and higher *ACS*. Although APF encourages cooperation, its *Overlap* is still higher than ACE, demonstrating ACE's superiority in discovering efficient cooperation strategies. Comparing ACE with MAPPO, which is trained in a synchronous manner, ACE demonstrates similar *ACS* to MAPPO with less *Time* and *Overlap*, which indicates the robustness of ACE to realistic execution with randomized action delay. Results of 3 agents can be found in appendix D.

6.4.2 ***Generalization to Agent Lost.*** We further consider another setting where the team size decreases within an episode on map size 25×25 to emulate the real-world scenarios with hardware failure and to examine whether our learned policies can generalize

[1]Maps in Habitat are harder than in the grid-based simulator, leading to differences in the success rate threshold.

Map Size	Methods	Synchronous Action Making				Asynchronous Action Making			
		Time ↓	Overlap ↓	Coverage ↑	ACS ↑	Time ↓	Overlap ↓	Coverage ↑	ACS ↑
15 × 15	Utility	40.81(0.94)	0.45(0.02)	**1.00(0.00)**	88.80(0.08)	35.75(0.99)	0.42(0.01)	**1.00(0.00)**	90.01(0.14)
	Nearest	25.44(0.53)	0.17(0.01)	**1.00(0.00)**	91.60(0.17)	22.59(0.34)	0.17(0.01)	**1.00(0.00)**	92.47(0.17)
	RRT	28.86(0.99)	0.18(0.01)	**1.00(0.00)**	91.46(0.03)	25.85(0.36)	0.19(0.01)	**1.00(0.00)**	92.36(0.08)
	APF	24.95(0.76)	0.17(0.01)	**1.00(0.00)**	91.57(0.39)	21.56(0.43)	0.17(0.01)	**1.00(0.00)**	92.52(0.37)
	Voronoi	48.57(3.64)	0.33(0.01)	**1.00(0.00)**	86.43(0.33)	42.94(4.05)	0.20(0.01)	**1.00(0.00)**	88.16(0.57)
	MAPPO	24.75(0.45)	0.08(0.02)	**1.00(0.00)**	92.39(0.19)	21.92(0.90)	0.09(0.01)	**1.00(0.00)**	93.18(0.17)
	ACE	**21.76(0.79)**	**0.07(0.00)**	1.00(0.00)	**92.54(0.21)**	**18.66(0.79)**	**0.07(0.01)**	1.00(0.00)	**93.39(0.14)**
25 × 25	Utility	189.38(0.93)	0.46(0.05)	0.93(0.01)	139.31(1.56)	183.71(1.59)	0.50(0.04)	0.95(0.01)	144.64(1.42)
	Nearest	113.48(1.73)	0.24(0.01)	**1.00(0.00)**	161.40(0.63)	99.52(2.00)	0.24(0.01)	**1.00(0.00)**	166.53(0.91)
	RRT	120.15(2.29)	0.20(0.01)	**1.00(0.00)**	164.35(0.64)	105.64(1.69)	0.21(0.01)	**1.00(0.00)**	168.33(0.27)
	APF	101.41(0.78)	0.23(0.01)	**1.00(0.00)**	162.46(0.64)	90.00(1.48)	0.23(0.01)	**1.00(0.00)**	166.82(0.81)
	Voronoi	131.65(0.41)	0.23(0.01)	**1.00(0.00)**	160.92(0.33)	117.14(0.13)	0.20(0.01)	**1.00(0.00)**	165.35(0.27)
	MAPPO	90.15(1.08)	0.08(0.01)	**1.00(0.00)**	168.54(0.58)	82.55(2.70)	0.09(0.01)	**1.00(0.00)**	171.72(0.27)
	ACE	**83.34(0.44)**	**0.06(0.00)**	**1.00(0.00)**	**170.03(0.49)**	**74.36(2.93)**	**0.06(0.00)**	**1.00(0.00)**	**173.16(0.72)**

Table 1: Performances of different methods under 2-agent synchronous and asynchronous settings in the grid-based simulator.

	Metrics	Utility	Nearest	RRT	APF	ACE
3 ⇒ 2	Time ↓	139.33(1.79)	76.53(1.16)	81.86(1.14)	74.80(2.11)	**67.50(1.42)**
	Overlap ↓	0.48(0.03)	0.30(0.01)	0.27(0.00)	0.32(0.01)	**0.22(0.00)**
	Coverage ↑	0.94(0.01)	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**
	ACS ↑	110.56(1.11)	126.03(0.39)	126.75(0.18)	125.15(0.58)	**128.49(0.37)**
4 ⇒ 3	Time ↓	96.15(0.46)	53.68(1.01)	55.33(0.82)	52.26(0.68)	**48.88(1.82)**
	Overlap ↓	0.40(0.03)	0.36(0.00)	0.34(0.01)	0.38(0.01)	**0.33(0.07)**
	Coverage ↑	0.92(0.01)	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**	**1.00(0.00)**
	ACS ↑	73.28(1.55)	84.05(0.57)	84.08(0.41)	83.60(0.41)	**84.78(0.77)**

Table 2: Performance of different methods with decreased team size on 25×25 maps in the grid-based simulator.

Methods	Utility	Nearest	RRT	APF	MAPPO	ACE
Time(s)	60.25(0.16)	38.72(0.12)	55.89(0.24)	52.64(0.23)	28.48(0.12)	**25.61(0.10)**

Table 3: Running time of different methods when the coverage ratio reaches 100% in the real-world robot system.

to these extreme cases during execution. "$N_1 \Rightarrow N_2$" denotes a scenario with N_1 agents at the beginning and only N_2 agents alive after 50% coverage. As shown in Table 2, ACE demonstrates 10% less *Time* than other baselines and obtains the highest *ACS* and lowest *Overlap*, indicating ACE's effective zero-shot adaptation to extreme situations where some agents go offline.

6.4.3 ***Real-World Robot System.*** In this part, we present the running time of different methods with 2 agents in real-world exploration tasks on 15×15 maps, which are running in an asynchronous manner. The deployment pipeline is described in Sec. 5.1. As shown in Table 3, two RL-based methods, MAPPO and ACE, outperform the planning-based baselines with a large margin according to the total exploration time. In particular, ACE reduces 33.86% real-world exploration time than the fastest planning-based method *Nearest*. Besides, ACE reduces 10.07% running time compared with MAPPO, proving that combining action-delay randomization with Async-MAPPO indeed improves the efficiency of multi-agent exploration.

6.5 Habitat Results

6.5.1 ***Main Results.*** We extend ACE to a vision-based environment, Habitat. Table 4 shows the performance of different methods under 2-agent asynchronous action-making settings. Despite having higher *Overlap* due to more exhaustive exploration, ACE outperforms planning-based baselines with ≥ 28% less *Time*, higher *Coverage* and *ACS*. Compared with synchronous MAPPO, ACE still shows higher *Coverage* and *ACS* with less *Time*, demonstrating the effectiveness of ACE in more complicated vision-based tasks.

6.5.2 ***Generalization to Agent Lost.*** We also consider the setting of decreased team sizes in Habitat, and we follow the same experimental setup as for the grid-based simulations. Table 5 shows the performance of different methods with decreased team size (2 ⇒ 1). ACE demonstrates 5.3% less *Time* than other baselines and obtains the highest *Coverage* and *ACS* with comparable *Overlap*, which indicates the ACE's ability to generalize to agent lost.

Methods	Time ↓	Overlap ↓	Coverage ↑	ACS ↑
Utility	273.83(37.80)	0.84(0.03)	0.83(0.08)	186.17(17.43)
Nearest	220.25(30.23)	**0.59(0.04)**	0.94(0.03)	180.05(7.37)
RRT	177.29(17.16)	0.63(0.06)	0.97(0.02)	187.35(6.86)
APF	218.45(24.64)	0.67(0.04)	0.94(0.02)	188.62(7.67)
MAANS	133.23(17.72)	0.68(0.09)	0.97(0.01)	201.33(7.98)
ACE	**127.62(8.55)**	0.78(0.07)	**0.98(0.01)**	**213.81(9.33)**

Table 4: Performance of different methods under 2-agent asynchronous action-making settings in Habitat.

Methods	Time ↓	Overlap ↓	Coverage ↑	ACS ↑
Utility	281.09(32.25)	0.48(0.05)	0.84(0.08)	153.02(12.38)
Nearest	309.76(9.83)	0.40(0.03)	0.85(0.05)	149.43(4.55)
RRT	260.31(27.92)	**0.35(0.03)**	**0.92(0.02)**	155.23(6.04)
APF	309.88(6.63)	0.42(0.01)	0.79(0.01)	143.54(0.83)
MAANS	262.92(19.84)	**0.35(0.04)**	0.90(0.03)	160.90(6.68)
ACE	**246.38(19.26)**	0.36(0.03)	**0.92(0.03)**	**164.32(8.23)**

Table 5: Performance of different methods with decreased team size in Habitat.

6.6 Ablation Studies

In this section, we analyze the sensitivity of communication size and action-delay randomization based on the grid-like simulator through ablation studies.

6.6.1 ***Sensitivity Analysis of Communication Size.*** We study the exploration performances in different communication traffic scenarios, including:

- **No Comm.**: The attention-based relation encoder is removed. Therefore, agents can only use their own local information to perform macro actions. This is the lower bound of different communication traffic.
- **Comm. (0.25x)**: The number of channels output by the CNN local feature extractor is set to 1, which is a quarter of the original 4 channels.
- **Comm. (0.5x)**: The number of CNN local feature extractor output channels is set to 2.
- **Perf. Comm.**: Agents use merged observation from all the agents as the input of the CNN local feature extractor.

Table 6 summarizes the performances on different communication traffic with 2 agents on 25×25 maps. More communication between agents generally leads to better exploration efficiency, as is shown by the decreasing *Time* and increasing *ACS* from "No Comm." to "Comm. (0.25x)", "Comm. (0.5x)" and "Perf. Comm.". Moreover, the behavior metric *Overlap* in these four scenarios shows better cooperation efficiency with more communication. Note that ACE performs even better than "Perf. Comm." with strictly less communication, demonstrating the effectiveness of the feature embedding extracted from our CNN policy for decision-making.

Methods	Time ↓	Overlap ↓	Coverage ↑	ACS ↑
No Comm.	159.26(2.18)	0.37(0.01)	0.93(0.01)	151.87(1.82)
Comm. (0.25x)	110.92(1.33)	0.11(0.01)	0.99(0.00)	167.60(0.71)
Comm. (0.5x)	83.77(1.38)	0.09(0.00)	**1.00(0.00)**	170.90(0.60)
Perf. Comm.	75.62(0.84)	**0.06(0.01)**	**1.00(0.00)**	173.15(0.53)
ACE	**74.36(2.93)**	**0.06(0.00)**	**1.00(0.00)**	**173.16(0.72)**

Table 6: Performance with different communication traffic.

Intervals	Time ↓	Overlap ↓	Coverage ↑	ACS ↑
Rand (1-10)	60.24(0.35)	0.51(0.00)	**1.00(0.00)**	82.44(0.31)
Rand (5-10)	56.41(1.51)	0.47(0.01)	**1.00(0.00)**	83.46(0.35)
Rand (1-5)	55.69(1.23)	0.43(0.01)	**1.00(0.00)**	83.57(0.45)
ACE (3-5)	**48.88(1.82)**	**0.33(0.07)**	**1.00(0.00)**	**84.78(0.77)**

Table 7: Performance of different action-delay intervals.

6.6.2 ***Sensitivity Analysis of Action-Delay Randomization.*** We further study the impact of the different random action-delay intervals. Besides the randomization interval stated in Sec. 4.2, we consider 3 different choices of action-delay intervals during training, "Rand (1-10)", "Rand (5-10)", and "Rand (1-5)". "Rand $(M_1 - M_2)$" means each macro action execution is delayed for a random number of simulation steps uniformly sampled from $[M_1, M_2]$. We empirically find that these variants have similar performance in most simple test settings, while ACE outperforms them in some extreme cases. To better illustrate the effect of different action-delay choices, we present the results in the "4 ⇒ 3" setting, an extreme scenario with agent loss. As shown in Table 7, ACE consumes the least *Time* and achieves the highest *ACS*. The results show that action-delay randomization works best with a proper randomization interval, while a large randomization interval adds high uncertainty during training and hurts the final performance.

7 CONCLUSION

To bridge the gap between synchronous simulator and asynchronous action-making process in real-world multi-agent exploration task, we propose a novel real-world multi-robot exploration solution, *Asynchronous Coordination Explorer (ACE)* to tackle this challenge. In ACE, Multi-agent PPO (MAPPO) is extended to the asynchronous action-making setting for effective training, and an action-delay-randomization technique is applied for better generalization to the real world. Besides, each agent equipped with a team-size-invariant Multi-tower-CNN-based Policy (MCP), extracts and broadcasts the low-dimensional feature embedding to accomplish efficient intra-agent communication. Although we aim at the sim-to-real problem caused by multiple agents executing tasks asynchronously, There are still many issues that have not been fully considered, such as communication errors, localization errors, and sensor errors. we leave these issues as our future work.

ACKNOWLEDGMENT

This research was supported by National Natural Science Foundation of China (No.U19B2019, 62203257, M-0248), Tsinghua University Initiative Scientific Research Program, Tsinghua-Meituan Joint Institute for Digital Life, Beijing National Research Center for Information Science, Technology (BNRist), and Beijing Innovation Center for Future Chips and 2030 Innovation Megaprojects of China (Programme on New Generation Artificial Intelligence) Grant No. 2021AAA0150000.

REFERENCES

[1] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. 2017. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles* 2, 3 (2017), 194–220.

[2] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. 2005. Coordinated multi-robot exploration. *IEEE Transactions on robotics* 21, 3 (2005), 376–386.

[3] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. 2020. Learning to explore using active neural slam. In *International Conference on Learning Representations*. ICLR.

[4] Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. 2021. Delay-aware model-based reinforcement learning for continuous control. *Neurocomputing* 450 (2021), 119–128. https://doi.org/10.1016/j.neucom.2021.04.015

[5] Tao Chen, Saurabh Gupta, and Abhinav Gupta. 2019. Learning exploration policies for navigation. In *International Conference on Learning Representations*. ICLR.

[6] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

[7] William W Cohen. 1996. Adaptive mapping and navigation by teams of simple robots. *Robotics and autonomous systems* 18, 4 (1996), 411–434.

[8] Christian Dornhege and Alexander Kleiner. 2013. A frontier-void-based approach for autonomous exploration in 3d. *Advanced Robotics* 27, 6 (2013), 459–468.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[10] Hangtian Jia, Yujing Hu, Yingfeng Chen, Chunxu Ren, Tangjie Lv, Changjie Fan, and Chongjie Zhang. 2020. Fever basketball: A complex, flexible, and asynchronized sports game environment for multi-agent reinforcement learning. *arXiv preprint arXiv:2012.03204* (2020).

[11] Miguel Juliá, Arturo Gil, and Oscar Reinoso. 2012. A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. *Autonomous Robots* 33, 4 (2012), 427–444.

[12] Alexander Kleiner, Johann Prediger, and Bernhard Nebel. 2006. RFID technology-based exploration and SLAM for search and rescue. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 4054–4059.

[13] Qian Long, Zihan Zhou, Abhinav Gupta, Fei Fang, Yi Wu, and Xiaolong Wang. 2020. Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning. In *International Conference on Learning Representations*.

[14] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. 2021. Learning high-speed flight in the wild. *Science Robotics* 6, 59 (2021), eabg5810. https://doi.org/10.1126/scirobotics.abg5810 arXiv:https://www.science.org/doi/pdf/10.1126/scirobotics.abg5810

[15] Shayegan Omidshafiei, Ali-akbar Agha-mohammadi, Christopher Amato, and Jonathan P. How. 2015. Decentralized Control of Partially Observable Markov Decision Processes using Belief Space Macro-actions. *CoRR* abs/1502.06030 (2015). arXiv:1502.06030 http://arxiv.org/abs/1502.06030

[16] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. 2018. Learning Dexterous In-Hand Manipulation. *CoRR* abs/1808.00177 (2018). arXiv:1808.00177 http://arxiv.org/abs/1808.00177

[17] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. 2017. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *CoRR* abs/1710.06537 (2017). arXiv:1710.06537 http://arxiv.org/abs/1710.06537

[18] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. 2019. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems* 16, 2 (2019), 1729881419839596.

[19] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. 2019. Episodic curiosity through reachability. In *International Conference on Learning Representations*. ICLR.

[20] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. 2019. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9339–9347.

[21] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 23–30.

[22] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. *CoRR* abs/1703.06907 (2017). arXiv:1703.06907 http://arxiv.org/abs/1703.06907

[23] Jaden B. Travnik, Kory W. Mathewson, Richard S. Sutton, and Patrick M. Pilarski. 2018. Reactive Reinforcement Learning in Asynchronous Environments. *Frontiers in Robotics and AI* 5 (2018). https://doi.org/10.3389/frobt.2018.00079

[24] Hassan Umari and Shayok Mukhopadhyay. 2017. Autonomous robotic exploration based on multiple rapidly-exploring randomized trees. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1396–1402. https://doi.org/10.1109/IROS.2017.8202319

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc.

[27] Lukas von Stumberg, Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. 2017. From monocular SLAM to autonomous drone exploration. In *2017 European Conference on Mobile Robots (ECMR)*. IEEE, 1–8.

[28] Haiyang Wang, Wenguan Wang, Xizhou Zhu, Jifeng Dai, and Liwei Wang. 2021. Collaborative Visual Navigation. *arXiv preprint arXiv:2107.01151* (2021).

[29] Tonghan Wang*, Jianhao Wang*, Yi Wu, and Chongjie Zhang. 2020. Influence-Based Multi-Agent Exploration. In *International Conference on Learning Representations*.

[30] Weixun Wang, Tianpei Yang, Yong Liu, Jianye Hao, Xiaotian Hao, Yujing Hu, Yingfeng Chen, Changjie Fan, and Yang Gao. 2020. From Few to More: Large-Scale Dynamic Multiagent Curriculum Learning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 7293–7300.

[31] Kai M Wurm, Cyrill Stachniss, and Wolfram Burgard. 2008. Coordinated multi-robot exploration using a segmentation of the environment. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1160–1165.

[32] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. 2018. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9068–9079.

[33] Yuchen Xiao, Weihao Tan, and Christopher Amato. 2022. Asynchronous Actor-Critic for Multi-Agent Reinforcement Learning. https://doi.org/10.48550/ARXIV.2209.10113

[34] Brian Yamauchi. 1997. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*. IEEE, 146–151.

[35] Jiachen Yang, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Hongyuan Zha. 2020. CM3: Cooperative Multi-goal Multi-stage Multi-agent Reinforcement Learning. In *International Conference on Learning Representations*.

[36] Meng Yao, Qiyue Yin, Jun Yang, Tongtong Yu, Shengqi Shen, Junge Zhang, Bin Liang, and Kaiqi Huang. 2021. The Partially Observable Asynchronous Multi-Agent Cooperation Challenge. *arXiv preprint arXiv:2112.03809* (2021).

[37] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. 2021. The surprising effectiveness of mappo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955* (2021).

[38] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. 2021. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games. https://doi.org/10.48550/ARXIV.2103.01955

[39] Chao Yu, Xinyi Yang, Jiaxuan Gao, Huazhong Yang, Yu Wang, and Yi Wu. 2021. Learning Efficient Multi-Agent Cooperative Visual Exploration. *arXiv preprint arXiv:2110.05734* (2021).

[40] Jincheng Yu, Jianming Tong, Yuanfan Xu, Zhilin Xu, Haolin Dong, Tianxiang Yang, and Yu Wang. 2021. SMMR-Explore: SubMap-based Multi-Robot Exploration System with Multi-robot Multi-target Potential Field Exploration Method. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*.

[41] Fengda Zhu, Siyi Hu, Yi Zhang, Haodong Hong, Yi Zhu, Xiaojun Chang, and Xiaodan Liang. 2021. MAIN: A Multi-agent Indoor Navigation Benchmark for Cooperative Learning. (2021).