
ASYNCHRONOUS COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING WITH LIMITED COMMUNICATION

A PREPRINT

Sydney Dolan

Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA 02139
sydneyd@mit.edu

Siddharth Nayak

Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA 02139
sidnayak@mit.edu

Jasmine Jerry Aloor

Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA
jjaloor@mit.edu

Hamsa Balakrishnan

Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA
hamsa@mit.edu

ABSTRACT

We consider the problem setting in which multiple autonomous agents must cooperatively navigate and perform tasks in an unknown, communication-constrained environment. Traditional multi-agent reinforcement learning (MARL) approaches assume synchronous communications and perform poorly in such environments. We propose AsynCoMARL, an asynchronous MARL approach that uses graph transformers to learn communication protocols from dynamic graphs. AsynCoMARL can accommodate infrequent and asynchronous communications between agents, with edges of the graph only forming when agents communicate with each other. We show that AsynCoMARL achieves similar success and collision rates as leading baselines, despite 26% fewer messages being passed between agents.

1 Introduction

Communication is crucial in cooperative multi-agent systems with partial observability, as it enables a better understanding of the environment and improves coordination. In extreme environments such as those underwater or in space, the frequency of communication between agents is often limited [1, 2]. For example, a satellite may not be able to reliably receive and react to messages from other satellites synchronously due to limited onboard power and communication delays. In these scenarios, agents aim to establish a communication protocol that allows them to operate independently while still receiving sufficient information to effectively coordinate with nearby agents.

Multi-agent reinforcement learning (MARL) has emerged as a popular approach for addressing cooperative navigation challenges involving multiple agents. Reinforcement learning approaches learn policies directly by interacting with a simulated environment for policy improvement. Unlike planning-based solutions [3], which require non-trivial implementation heuristics and expensive inference computation at execution time, reinforcement learning offers a way to represent complex strategies with minimal overhead once the policies are trained. The classical MARL formulation follows a synchronous framework, where all agents take actions simultaneously, with actions executed immediately at each time step. Similarly, communications between agents [are assumed to] occur instantaneously, frequently, and synchronously, often with agents broadcasting their state to every other agent in the environment. As a result, traditional MARL algorithms are poorly suited to asynchronous settings where agents operate on independent time scales and cannot frequently communicate with one another. While prior work has explored how to coordinate agents using less communication, differences in message encoding approach can significantly impact performance, as shown in Section 5. Furthermore, little attention has been given to achieving such coordination asynchronously. Unfortunately, existing

asynchronous approaches often increase communication to compensate for the lack of synchronized coordination among agents. Our work builds on prior work in multi-agent communication by introducing an asynchronous MARL framework that enables agents to minimize communication while completing navigation tasks. The main contributions of this paper are:

1. We propose AsynCoMARL, a graph transformer-based communication protocol for MARL that relies on dynamic graphs to capture asynchronous and infrequent communications between agents.
2. We empirically evaluate AsynCoMARL on two MARL benchmarks (Cooperative Navigation [4] and Rover-Tower [5]) and show that our method can achieve superior performance while using less communication.

AsynCoMARL is an asynchronous MARL approach that leverages graph transformers to learn communication protocols from dynamic graphs. In this setting, agents seek to learn the communication protocol that best utilizes asynchronous and infrequent communications from nearby agents. Each agent’s graph transformer utilizes a dynamic weighted directed graph to learn a communication protocol with other active agents in its vicinity. The underlying MARL algorithm uses this learned graph transformer encoding in both the actor and the critic to optimize action selection, leading to effective cooperation with less communication between agents. We conduct experiments in two environments, Cooperative Navigation and Rover-Tower, chosen to replicate the communication-constrained settings of space missions and planetary rover exploration. We find that strategies learned by AsynCoMARL use less communication and outperform other MARL methods.

2 Related Work

2.1 Attention and Graph-Based Methods for Multi-Agent Communication

Through communication, agents can obtain a better understanding of their own environment and other agents’ behaviors, thus improving their ability to coordinate. For a comprehensive survey on research on the impact of communication in multi-agent collaboration, we refer the reader to [6]. Seminal works such as CommNet [7] used a shared neural network to process local observations for each agent. Each agent makes decisions based on its observations and a mean vector of messages from other agents. Although agents can operate independently with copies of the shared network, instantaneous communication with all agents is necessary. Subsequent works [8, 9] aimed to investigate methodologies that improved coordination by using a small subset of agents. In ATOC [9], the authors used a probabilistic gate mechanism to communicate with nearby agents in an observable field and a bi-LSTM to concatenate messages together. Further improvement to multi-agent collaboration was made by learning encodings of local information via attention mechanisms. TarMAC [10] uses an attention mechanism to generate encodings of the content of messages so that nearby agents can learn the message’s significance via a neural network implicitly. While attention-based mechanisms can help agents improve their abilities to utilize individual pieces of communication, attention-based approaches neglect larger inter-agent relationships. Graph-based methods like DGN [11], DICG [12], InforMARL [13], MAGIC [14], and EMP [15] formulate interactions between different agents as a graph. These methods use graph structures to learn relational representations from nearby nodes. However, a similar communication problem to CommNet arises with these methodologies, as they often require fully connected graphs during the learning process. For example, in EMP, all agents must know the position of all other entities in the graph at the beginning of an episode. This assumption is a key limitation, as it requires that all agents will be able to coordinate synchronously.

2.2 Event-Triggered Communication

Event-triggered control is a control strategy in which the system updates its control actions only when certain events or conditions occur, rather than continuously or at regular time intervals [16]. By reducing the number of control updates, event-triggered control can significantly decrease the workload on controllers, a beneficial attribute for systems with limited resources. ETC [17], VBC [18], and MBC [19] have proposed event-triggered control methodologies to reduce the communication frequency and address communication constraints. These works focus on optimizing when transmission can occur rather than optimizing how to achieve better performance with less communication. Menda et al. [20] frame the asynchronous decision-making problem where agents choose actions when prompted by an event or some set of events occurring in the environment. By relying on a continuous state-space representation and an event-driven simulator, the agents step from event to event, and lower-level timestep simulations are eliminated. This improves the scalability of the algorithm but does not capture low-level agent-agent interactions during an event.

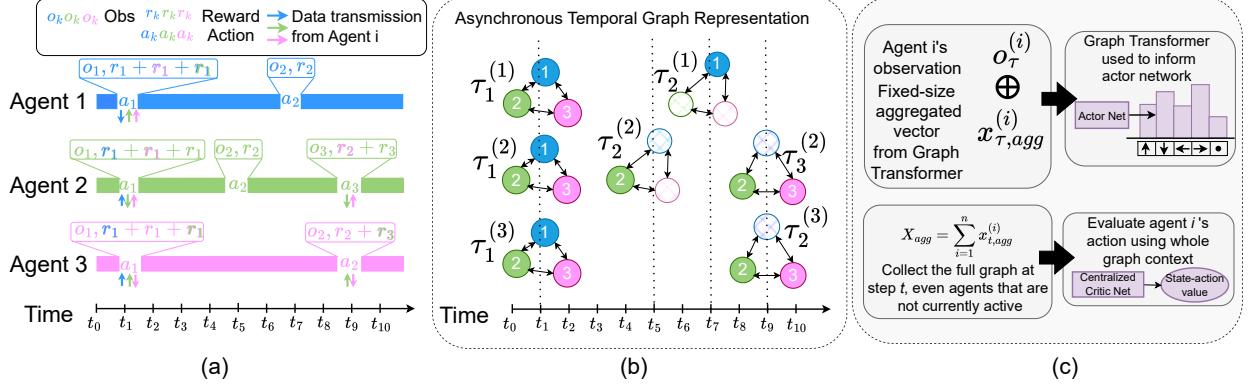


Figure 1: Overview of AsynCoMARL: (a) Environment. Agents within our environment take actions and observations asynchronously. To encourage collaboration, when agents take actions at the same time t , they receive a shared reward. The sequence of actions and observations for agent i is referred to by timescale $\tau^{(i)}$. The arrows indicate data transmissions, which represent the most recent graph observation $x_{\tau_{agg}}^{(i)}$. b) Asynchronous Temporal Graph Representation. Each active agent within our environment is translated to become a node on the graph, and they can communicate with other agents located nearby within distance ϕ . Our graph representation is dynamic, meaning that graph edges connect and disconnect depending on agent proximity. c) Agent i 's observation is combined with its node observations from the graph transformer, $x_{\tau_{agg}}^{(i)}$, and fed into the actor network. The critic takes the full graph representation X_{agg} and evaluates agent i 's action.

2.3 Asynchronous Actor Critic

An alternative approach to asynchronous environments formulates the problem as a Macro-Action Decentralized Partially Observable Markov Decision Process (MacDec-POMDP) [21], where agents can start and end higher level 'macro' actions at different time steps. Xiao et al. [22] propose a methodology for multi-agent policy gradients that allow agents to asynchronously learn high-level policies over a set of pre-defined macro actions. The method relies on an independent actor independent centralized critic training framework, named IAICC, to train each agent. Hong et al. [23] alter the IAICC framework to encode agent time history independently to avoid duplicate macro-observations in the centralized critic. These works focus on learning algorithms for asynchronous *macro-actions* that occur across multiple time steps. By contrast, our work focuses on the micro level, looking for planning opportunities while agents achieve the same macro-level task.

3 Preliminaries

3.1 Decentralized Partially Observable Markov Games

A decentralized partially observable Markov decision process (DEC-POMDP) is a multi-agent extension of a Markov decision process (MDP) where multiple players (agents) interact in a shared environment, but each agent has only limited or partial information about the current state of the environment and the state of other agents. A DEC-POMDP for N agents can be defined by a set of global states \mathcal{S} , a set of private observations for each agent $o^{(1)}, o^{(2)}, \dots, o^{(n)}$, a set of actions for each agent, $a^{(1)}, a^{(2)}, \dots, a^{(n)}$, and the transition function $T : \mathcal{S} \times a^{(1)} \times \dots \times a^{(N)} \rightarrow \mathcal{S}$. Agent i chooses actions $a^{(i)} \in \mathcal{A}^{(i)}$, and then obtains a reward as a function of its state-action pairing: $r^{(i)} : \mathcal{S} \times a^{(i)} \rightarrow \mathbb{R}$. It also receives a local observation $o^{(i)}$. Agent i aims to maximize its reward $R^{(i)} = \sum_{t=0}^T r^{(i)}$.

3.2 Policy Gradient Methods

The policy gradient method is used in RL tasks to perform gradient ascent on the agent policy parameters, θ , to optimize the total discounted reward, $J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[R]$. Here, ρ^π is the state distribution, π_θ is the policy distribution and R represents the reward at that time t , $R_t = \sum_{t'=t}^T r(s_{t'}, a_{t'})$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} \left[\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t | s_t) R_t \right] \quad (1)$$

In lieu of R_t , we use the advantage function $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ to decrease the variance of the estimated policy gradient, where $V^\pi(s_t)$ is the value function.

We adopt a centralized training decentralized execution learning paradigm [24, 25]. In execution, agents learned policies are conditioned only on their own action-observation history. In training, the critic has access to the global state \mathcal{S} of the environment, which is used to train the model end to end.

4 Methodology

In this section, we introduce our proposed multi-agent graph transformer communication protocol, AsynCoMARL. We consider a partially observable setting of N agents, where agent i receives local observation o_i^t at time t , containing local information from the global state \mathcal{S} . The agent, i , learns a communication-based policy π_i to output a distribution over actions $a_t^{(i)} \sim \pi^{(i)}$ at time step t . Here, we present a description of our graph formulation and its integration into the graph transformer, a description of our framework’s key components, and our training procedure.

4.1 Asynchronous Formulation

In traditional multi-agent reinforcement learning, agents always take action steps synchronously without considering the communication constraints associated with their action selection. For instance, when exploring unknown environments, a single rover may navigate to an area where they are unable to communicate with others, or the duration of transmitting a single message may take several time steps. To model these communication-action costs, we define a new time scale τ to account for the specific actions each agent has taken at different time steps. As shown in Figure 1, $\tau_1^{(i)}$ represents the time of the first action that agent i has taken. In Figure 1 panel (b), all three agents take their first action at the same t , resulting in the same time reference point for τ_1 . However, the next time t that each agent takes their next action is different (agent 1 $\tau_2^{(1)} = t_7$, agent 2 $\tau_2^{(2)} = t_5$ agent 3 $\tau_2^{(3)} = t_9$). Rather than incorporating all time steps into the replay buffer, we only include those steps where the agents are taking an action. As a result, the replay buffer of each agent’s trajectory is based on their τ sequence of actions instead of t . To improve the generalizability of our algorithm to different periods of time between actions τ_1 and τ_2 , during training, we randomly generate the period between different actions. We refer to this randomized parameter as μ , and it determines when an agent will take a subsequent action. μ is chosen from a uniform distribution, with different intervals used for training and testing to ensure diversity. The μ range is kept relatively small to model asynchronous behavior that maintains staggered interactions between agents without diverging to entirely independent timescales. For each initiation, the delay is selected randomly from this distribution and remains constant throughout the episode rather than changing after specific actions. This decision avoids creating an artificial link between certain movement actions and specific delays, as we consider such realism beyond the scope of this work and more applicable to domain-specific problems.

4.2 Graph Overview

Graph transformers are a specialized type of transformer model designed to handle graph-structured data. In addition to the traditional transformer’s ability to capture long-range dependencies, graph transformers enable the model to understand both local node interactions via graph edges and global context via self-attention.

As an input into our graph transformer, we form an agent-entity graph [15], where every object in the environment is assumed to be either an agent, an obstacle, or a goal. All objects within the environment are transformed to be nodes on the graph, with node features $x_j = [p_i^j, v_i^j, p_i^{\text{goal},j}, \text{entity_type}(j)]$ where $p_i^j, v_i^j, p_i^{\text{goal},j}$ are the *relative* position, velocity, and position of the goal of the object at node j with respect to agent i , respectively. If node j corresponds to a (static/dynamic) obstacle or a goal, we set $p_i^{\text{goal},j} \equiv p_i^j$. To process the *entity_type* categorical variable, we use an embedding layer.

Our graph formulation is dynamic, meaning that edges are formed depending on an entity’s proximity to other objects in the graph and their corresponding communication interval, as shown by (b) in Figure 1. We define an edge to exist $e \in \mathcal{E}$ between an agent and another agent if they are within a ‘communication radius’ λ of each other and if they are taking an action at the same time step t . Edges are formed between agents and obstacles or landmarks when they are merely within the agent’s communication radius. As landmarks and obstacles do not have decision-making abilities, their presence can be sensed at every time step the agent is in proximity. When an edge is formed, e_{ij} , it has an associated edge feature given by the Euclidean distance between the entities involved, i and j .

For a complete representation of our graph structure, we create an adjacency matrix to represent the connectivity between different nodes on a graph. An adjacency matrix \mathbb{A} is a square matrix used to describe the connections between

nodes on a graph, where each element $\mathbb{A}_{i,j}$ represents the presence or absence of an edge between node i and node j . Our graph is directed and weighted, so the value in the adjacency matrix for $\mathbb{A}_{i,j}$ is proportionate to the weight of the edge between nodes i and j . To account for the fact that our agents can take actions at different time steps, we first introduce the variable d_i to represent the current status of the agent, where $d_i = 1$ if the agent is active and $d_i = 0$ if the agent is inactive. We create an additional matrix to reflect the activity status of all nodes in the graph $\mathcal{D} \in \mathbb{R}^{N \times N}$ where $\mathcal{D}[i, j] = d_i \cdot d_j$, where d_i and d_j reflect whether node i and node j are active. This is used to ensure that interactions can only occur if both nodes are active. The masked adjacency matrix $\mathbb{A}^{\text{masked}}$ is given by the element-wise product $\mathbb{A} \circ \mathcal{D}$. The resulting masked adjacency matrix dynamically updates as agents become inactive, allowing the algorithm to focus on interactions between remaining agents. Similarly, a death mask enforces that no interactions or learning updates are computed for agents whose tasks are already completed.

4.3 Graph Transformer

We rely on a graph transformer to encode messages and characterize relationships between different entities in the environment. Our transformer takes the node features x_k and edge features e_k as input. The graph transformer is based on a Unified Message Passing Model (UniMP) [26] that relies on multi-head dot product attention to selectively prioritize incoming messages from their neighbors based on their relevance. We rely on two layers of this model, with each layer update defined as

$$x'_i = W_1 \cdot x_i + \sum_{j \in \mathbb{N}(i)} \alpha_{i,j} W_2 \cdot x_j \quad (2)$$

where x_k are the node features in the graph, $\mathbb{N}(i)$ is the set of nodes which are connected to node i , W_k are learnable weight matrices and the attention coefficients. $\alpha_{i,j}$ is computed via multi-head dot product attention

4.4 Reward Structure

At every time step, each agent gets a distance-based reward to an assigned goal, $\mathcal{R}_{\text{dist}}(s_\tau, a_\tau^{(i)})$. When an individual agent, i , reaches their assigned goal (indicated by ρ), it receives a goal-reaching reward $\mathcal{R}_{\text{goal}}(s_\tau, a_\tau^{(i)})$. ρ is 1 if the agent reached the assigned goal and was previously not at the assigned goal; otherwise, 0. We also penalize agents colliding with other agents or obstacles in the environment using a collision penalty $-C$. κ is a 0/1 variable that indicates if an agent collided with another agent or an obstacle. The reward for agent i at time step t then becomes

$$\mathcal{R}_\tau^{(i)}(s_\tau^{(i)}, a_\tau^{(i)}) = \mathcal{R}_{\text{dist}}(s_\tau^{(i)}, a_\tau^{(i)}) + \rho \mathcal{R}_{\text{goal}}(s_\tau^{(i)}, a_\tau^{(i)}) - \kappa C \quad (3)$$

Rewards are shared between all agents active at step t . The purpose of sharing rewards between active agents is to encourage synchronous collaboration when possible.

$$\mathcal{R}_{\text{total}}(s_\tau, A_\tau) = \sum_i^{\eta_{act}} \mathcal{R}_\tau^{(i)}(s_\tau^{(i)}, a_\tau^{(i)}) \quad (4)$$

4.5 Training

For our algorithm, the traditional DEC-POMDP setup is augmented by the existence of the graph network, transforming the tuple that describes the problem to: $(\mathcal{N}, \mathcal{S}, \mathcal{O}, \mathcal{A}, R, P, \gamma, \mathcal{G})$. $g^{(i)} = \mathcal{G}(s; i)$ represents the graph network formed by the entities of the environment with respect to agent i . Features of the graph structure are used in the policy gradient learning architecture.

Our learning architecture relies on a centralized critic, which uses the state-action pairs and information from the graph formulation. As shown in Figure 1, the critic receives the full graph embeddings via a global mean pooling operator ($X_{\text{agg}} = \frac{1}{N} \sum_{i=1}^N x_{\tau, \text{agg}}^{(i)}$) so that it has a global view of the graph representation. The critic updates parameter ϕ to approximate the value function.

$$V_\phi^{\pi_{\theta_i}}(s_\tau^{(i)}, a_\tau^{(i)}, X_{\text{agg}}) = \mathbb{E}_{a \sim \pi_{\theta_i}} \left[\sum_{\tau=0}^T \gamma^\tau \mathcal{R}_\tau | s_0 = s \right] \quad (5)$$

We then create an individual actor for each agent. Each agent relies on policy $\pi_{\theta_i}^{(i)}(a_\tau^{(i)} | o_\tau^{(i)}, g_\tau^{(i)})$, parameterized by θ_i to determine its action $a^{(i)}$ from its local observation $o^{(i)}$ and its local graph network $g^{(i)}$. The resultant policy gradient is:

Algorithm 1 AsynCoMARL Training setup

```

1: Initialize a decentralized policy network for each agent  $i$ :  $\vec{\pi}_{\theta_i}$ 
2: Initialize a centralized critic network  $V_{\phi}^{\vec{\pi}_{\theta}}$ 
3: for  $episode = 1 : M$  do
4:   Reset the environment
5:   Reset buffer  $\mathcal{D} = \{\}$ 
6:   for step  $t = 0 : T_f$  do
7:     if  $\exists i \in \{1, \dots, n\}$   $agent^{(i)}.status = active$  then
8:        $a^{(i)} \leftarrow$  get actions of active agents
9:       Update environment state with  $a^{(i)}$ 
10:       $\mathcal{R}_{total}^{(i)} \leftarrow \sum_{j=1}^{\eta_{active}} \mathcal{R}_t^{(j)}$ 
11:      Collect  $\langle o^{(i)}, a^{(i)}, g^{(i)}, r^{(i)} \rangle$  into buffer  $\mathcal{D}$  at  $\tau^{(i)}$ 
12:    else
13:      for  $i \in \{1, \dots, n\}$  do
14:         $a^{(i)} = \emptyset$ 
15:        Update environment state with  $a^{(i)}$ 
16:         $mask(i) = \begin{cases} 1, & \text{if agent } i \text{ is finished} \\ 0, & \text{otherwise} \end{cases}$ 
17:      end for
18:    end if
19:  end for
20:  Follow standard MARL update process from MAPPO [27]
21: end for

```

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\pi_{\theta_i}} [\nabla_{\theta_i} \log \pi_{\theta_i}(a_{\tau}^{(i)} | o_{\tau}^{(i)}, g_{\tau}^{(i)}) \cdot V_{\phi}^{\pi_{\theta_i}}(s_{\tau}^{(i)}, a_{\tau}^{(i)}, X_{agg})] \quad (6)$$

Algorithm 1 shows the complete training setup for AsynCoMARL. As mentioned in Section 4.1, each agent i has its own interval μ , which determines when it will take a subsequent action. The interval is reset for each environment scenario to avoid overfitting. During a training episode, an agent's status is marked as active if there have been μ steps since its last action. If an agent reaches its goal before the end of the episode, its status d_i is changed to 0, thus masking its presence in the masked adjacency matrix A^{masked} passed into the centralized critic function.

We rely on the update procedure associated with the MAPPO algorithm [27]. In this algorithm, the advantage estimate is computed using Generalized Advantage Estimation (GAE) method [28], with advantage normalization and value clipping over the complete trajectory. The value function is clipped to avoid large updates, ensuring robust learning. A random mini-batch is sampled from the data buffer, and for each data chunk in the mini-batch, the RNN hidden states for both the policy and the critic are initialized from the first hidden state in the data chunk. The policy and critic parameters are then updated using the Adam optimizer [29], based on the computed loss, which incorporates the policy objective, value loss, and entropy regularization for exploration.

5 Experimental Results

5.1 Experimental Setting

We conduct experiments in two environments, Cooperative Navigation and Rover-Tower, specifically chosen to emulate real-world scenarios with communication constraints. These two environments replicate communications encountered in space missions and planetary rover exploration.

The Cooperative Navigation environment was chosen because it simulates the space environment, a setting where communications are constrained and sporadic but where efficient coordination is important. We use the Cooperative Navigation environment to evaluate the amount of communication used in each approach. In particular, we aim to study how frequently our approach requires communications between agents compared to other baselines.

The Rover-Tower environment was chosen as it simulates a real-world planetary exploration scenario involving a rover and a more capable observing agent. We use the Rover-Tower environment to assess each method's ability to adapt to

more complex communication scenarios. In particular, we aim to study how generalizable each method is to different communication scenarios involving agents with different observation and communication abilities.

Both the Cooperative Navigation and Rover-Tower environments are implemented in the multi-agent particle environment (MPE) framework introduced by [30]. MPE involves a set of agents that have a discrete action space where it can apply a control unit acceleration in the x - and y - directions. We evaluate our proposed model in the following two environments:

Cooperative Navigation: Dolan et al. [4] propose a modified MPE environment that uses the satellite dynamics following the Chloessy-Wiltshire model [31], where a set of n satellites are moving within a 2D plane. Each satellite is tasked with rendezvousing to an assigned goal location by the end of the episode. The Chloessy-Wiltshire equations are a set of second-order differential equations whose x and y components are coupled, thereby increasing the difficulty of the simulation dynamics. We use this modified environment to test our experiment due to its relevancy to the problems we are interested in (e.g., environments with limited communications) and because the agents involved will continue along their relative trajectories even when not communicating with one another. In the Cooperative Navigation task, satellites are initialized into an environment defined by the world size (e.g., 2 km). We set the initialization such that each satellite is at least 500 meters away from all other satellites and their intended goals. This constraint ensures that satellites have sufficient space to maneuver, preventing the occurrence of deadlock. For each episode, each satellite is reinitialized at a different location with a new goal, ensuring a variety of maneuvers and enhancing the generalizability to different navigation interactions.

Rover-Tower: Iqbal and Sha [5] develop the Rover-Tower task environment where randomly paired agents communicate information and coordinate. The environment consists of 4 "rovers" and 4 "towers", with each episode pairing rovers and towers randomly. The performance of the pair is penalized on the basis of the distance of the rover to its goal. The task is designed to simulate scientific navigation on another planet where there is limited infrastructure and low visibility. Rovers cannot observe their surroundings and must rely on communication from the towers, which can locate rovers and their destination and can send one of 5 discrete communications to their paired rover. In this scenario, communication is highly restricted. Extended descriptions of both environments are included in the appendix.

5.2 Evaluation Metrics

We rely on several evaluation metrics to characterize the performance of our algorithm against other baselines. We selected these metrics to assess the agent's ability to successfully navigate to their goals without collisions.

- **Communication frequency (f_{comm}):** Average ratio of the number of messages passed between agents over the maximum number of communication opportunities in the episode. f_{comm} of 1 indicates each agent messaged every other agent at every time step in the episode (lower value indicates the model is more message efficient).
- **Success Rate ($S\%$):** Percentage of episodes in which all n agents are able to get to their goals (higher is better).
- **Fraction of Episode Completed (T):** The fraction of an episode that agents take on average to get to the goal. If the agents do not reach the goal, then the fraction is set to 1. For each episode, the n episode fractions are averaged together; this number is then normalized across all evaluation episodes (lower is better).
- **Average Collisions (#col):** the total number of collisions that agents had in an episode, normalized by the number of agents and then normalized by the number of evaluation episodes (lower is better).

While the standard reported metric for MPE environments is the global reward, we have not included it in our example results. Since we introduced a new reward formulation for our model, we found reward comparisons challenging to both interpret and compare, given the different reward functions that the baselines were designed with. Similar to [32], we use the success rate metric to indicate the overall success of the total number of agents.

5.3 Training Details

In the simulation, every policy is trained with 2M steps over 5 random seeds. All results are averaged over 100 testing episodes. Associated hyperparameters for each baseline are included in the appendix. All models are generated by running on a cluster of computer nodes on a Linux operating system. We use Intel Xeon Gold 6248 processors with 384GB of RAM.

Algorithm	$N = 3$				$N = 5$				$N = 7$				$N = 10$			
	$f_{\text{comm}} \downarrow$	$T \downarrow$	# col \downarrow	S% \uparrow	$f_{\text{comm}} \downarrow$	$T \downarrow$	# col \downarrow	S% \uparrow	$f_{\text{comm}} \downarrow$	$T \downarrow$	# col \downarrow	S% \uparrow	$f_{\text{comm}} \downarrow$	$T \downarrow$	# col \downarrow	S% \uparrow
GCS [33]	1.0	0.36	0.34	100	1.0	0.42	1.72	98	1.0	0.39	2.86	100	1.0	0.78	7.38	1
asyncMAPPO [34]	0.21	0.10	0.86	100	0.20	0.32	6.05	100	0.19	0.23	12.3	100	0.15	0.14	25.68	100
Actor-Attention-Critic [5]	0.21	0.42	0.30	100	0.16	0.47	1.20	100	0.11	0.49	2.52	100	0.08	0.52	4.2	100
TransfQmix [32]	0.13	0.83	0.02	42	0.16	0.96	0.12	39	0.17	0.96	0.28	33	0.18	0.96	0.12	19
CACOM [35]	0.26	0.99	0.17	0	0.12	0.97	0.35	0	0.12	0.97	0.87	0	0.10	0.98	1.46	0
DGN [11]	0.20	0.96	0.12	0	0.13	0.99	0.06	0	0.09	0.99	0.07	0	0.06	0.98	0.42	0
AsynCoMARL	0.10	0.24	0.45	97	0.08	0.23	0.85	98	0.08	0.25	2.16	86	0.05	0.34	6.38	86

Table 1: Comparison of AsynCoMARL with other baseline methods for scenarios with 3, 5, 7, and 10 agents in the Cooperative Navigation environment.

5.4 Comparison of AsynCoMARL with Other Methods

In this section, we demonstrate that AsynCoMARL can effectively learn policies for navigation even in settings with less frequent and asynchronous communications.

We compare our methodology against several alternative MARL frameworks that seek to provide limited communication.

- **GCS** [33]: The authors factorize the joint team policy into a graph generator and graph-based coordinated policy to enable coordinated behavior among agents.
- **Actor-Attention-Critic** [5]: Actor-Attention-Critic uses a centralized critic with an attention mechanism that dynamically selects which agents to attend to at any time point during training.
- **asyncMAPPO** [34]: The authors extend multi-agent proximal policy optimization [25] to the asynchronous setting and apply an invariant CNN-based policy to address intra-agent communication.
- **TransfQMIX** [32]: TransfQMIX relies on graph transformers to learn encodings over the state of observable entities, and then a multi-layer perceptron to project the q-values of the actions sampled by the individual agents over the q-value of the joint sampled action.
- **CACOM** [35]: The authors employ a two-stage communication scheme where agents first exchange coarse representations and then use attention mechanisms and learned step size quantization techniques to provide personalized messages for the receivers.
- **DGN** [11]: DGN relies on graph convolutional networks to model the relational representation and implicitly models the action coordination.

5.4.1 Performance on Cooperative Navigation Environment

Table 1 compares the performance of AsynCoMARL against the other baselines. Although having a small number of collisions is better, the policies of some of the baseline algorithms do not significantly move the agents from their initial positions after training and, hence, do not get to the goal. This leads to them having a lower number of collisions. Hence, this metric should be judged by the success rate in context. Similarly, the episode completion rate and communication frequency should be considered within the context of the overall success and collision rates.

When evaluating AsynCoMARL’s performance in the context of these other baselines, our method is able to achieve high success rates and relatively low collision rates, despite 26% fewer messages being passed between agents. The temporal graph formulation of our model, which inherently allows communications to be masked to reduce communication overhead during training, leads to a method capable of handling trade-offs between communication frequency, success, and collision avoidance.

When comparing AsynCoMARL against other baselines, there are immediate takeaways from the $n = 3$ agent case. GCS relies on an acyclic uni-directional graph representation that requires the most recent action selection at the step prior, resulting in high success rates at the cost of a significantly higher communication frequency. Both asyncMAPPO and Actor-Attention-Critic demonstrate comparable performance in success and collision rates for $n = 3$ agents. Similar to the design of AsynCoMARL, Actor-Attention-Critic is designed to dynamically select which agents to focus on. This reduces f_{comm} and leads to improved success and collision rates. However, this attention mechanism overlooks relationships between agents captured by the graph representation used in AsynCoMARL, leading Actor-Attention-Critic to have a higher communication frequency and episode completion rates.

The performance of TransfQMIX is comparatively less effective. TransfQMIX exhibits the lowest collision rates of the algorithms evaluated but at the cost of a low success rate. As stated previously, low collision rates should be considered in the context of the success rate and episode completion rate, as it is possible for agents to learn a policy in which they do not move at all.

Algorithm	Metrics		
	$f_{\text{comm}} \downarrow$	$T \downarrow$	$S\% \uparrow$
Actor-Attention-Critic [5]	0.21	0.84	56%
AsyncMAPPO [34]	0.24	0.98	0%
TransfQmix [32]	0.40	0.98	0%
AsynCoMARL (our method)	0.14	0.55	50%

Table 2: Comparison of AsynCoMARL with other baseline methods for scenarios with 4 rovers and 4 towers in the Rover-Tower environment.

Both DGN and CACOM fail to learn meaningful synthesis of agent communication and have poor success rates as a result. DGN relies on a graph convolutional network, where all neighboring agents contribute equally to the aggregation of node features. The poor performance of DGN in this setting suggests that equal weighting of nearby agents leaves agents unable to capture nuances in the graph structure that are captured through our agent-entity graph embeddings. As noted in CACOM, the learned gate-pruning contains relatively high variance in the Cooperative Navigation environment and is subjected to instability. We believe the complexity of the dynamics in our setting, coupled with the asynchronous formulation, resulted in learning instability in the gating function and, subsequently, poor performance.

When considering larger numbers of agents ($n = 5, 7, 10$), we see similar trends. CACOM and DGN continue to struggle to meaningfully encode information from nearby neighbors at scale. While asyncMAPPO maintains its strong performance (as evidenced by its success rate), it also possesses a significantly higher number of collisions than AsynCoMARL. The performance of Actor-Attention-Critic and AsynCoMARL are similar. However, the Actor-Attention critic approach requires more frequent communication between agents and results in more collisions in the $n = 5$ and $n = 7$ cases. As the number of agents increases to $n = 10$, we note that the performance of GCS suffers. Upon further inspection, we found that GCS could match the performance it had on the $n = 3, 5, 7$ cases with additional training time (e.g., 2 million steps vs. 5 million steps). This decrease in performance in GCS suggests that the fully connected graph feature of this model serves to increase computational training time and hinders the ease of scaling the model.

We note that with the increased number of agents, the communication frequency of all algorithms generally decreases. This can be attributed to the increase in world size, resulting in a less dense environment and fewer communications relative to the number of total communications that would be possible for $n = 10$ agents. As a result, for the $n = 10$ case, the communication frequencies are relatively low.

5.4.2 Performance on Rover-Tower Environment

Table 2 shows AsynCoMARL against the best-performing baselines from the prior experiment. As a reminder, the reward function associated with this environment does not include any collision penalty, so we do not include the $\#col$ metric. In this environment, rovers must rely on encoded messages from their corresponding tower to determine their action selection, whereas towers have more advanced observation abilities. To account for these two classes, Actor-Attention-Critic creates a separate network for the rover class and the tower class, whereas AsynCoMARL does not. Despite the fact that AsynCoMARL is using a singular network to represent both the rovers and the towers, it still achieves a comparable success rate to the Actor-Attention-Critic. Additionally, AsynCoMARL relies on less communication and produces faster episode completion rates than other baselines, suggesting that AsynCoMARL is a more efficient, generalizable communication protocol for this environment.

5.4.3 Visualizing the Graph Transformer Weights

To better understand the underlying mechanisms of our graph transformer communication protocol, we visualize the graph transformer attention weights at three different times in an episode for a single agent in the Cooperative Navigation environment. In Figure 2, the leftmost panel corresponds to the weights at time $\tau = 0$. In this panel, agent 0 is unconnected to any other agents, and thus, the attention weighting for all other nodes is perfectly equal. The center panel corresponds with $\tau = 6$ for agent 0, and at this point in the episode, agent 0 is now able to communicate with agent 3 due to their proximity. As a result, the attention weighting of the messages from agent 3 at this time is higher than the other agents, as shown by the attention weighting panel. The final rightmost panel corresponds with $\tau = 18$ for agent 0. At this point in the episode, the agent is now within the communication range of agents 1 and 4, as shown by the connecting edges in the agent locations panel. Similar to what we observed in the second panel, the corresponding attention weights to these two agents are also higher. Interestingly, we also find that the attention weight for agent 2 is also fairly large, despite not being connected via graph. Upon further inspection, we found that this is attributed to the

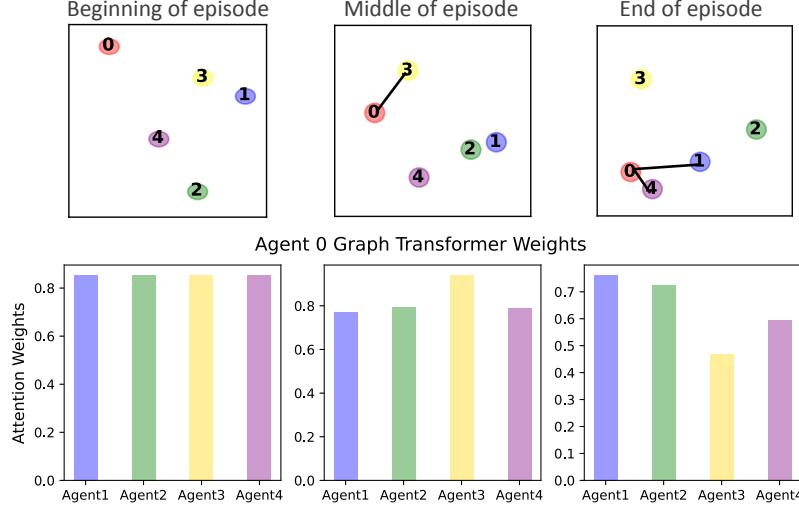


Figure 2: Attention weights for agent 0 in the $n = 5$ agent Cooperative Navigation task. We compare the changes in graph transformer attention at three discrete periods during the episode at the beginning, middle, and end.

communication frequencies of agents 0 and 2; both of these two agents communicated more frequently throughout the episode than agent 0 did with agents 3 and 4. We find that the graph transformer communication protocol learns to attend to both agents in proximity to the active agent (e.g., panel 2) but also to those agents from whom it gets more frequent communication (e.g., panel 3). Therefore, the model implicitly learns the trade-off between agent proximity and frequency of communication from specific agents.

5.5 Ablation Studies

5.5.1 Impact of Graph Transformer

To verify the effectiveness of our graph transformer communication protocol, we conduct an ablation study on the Cooperative Navigation environment. We train and evaluate two models on $n = 10$ agents: (1) AsynCoMARL, our graph transformer-based communication protocol for multi-agent reinforcement learning, and (2) MARL, our stripped-down asynchronous multi-agent reinforcement learning formulation that only communicates when agents are active at the same time step. We compare the differences between the two models in Table 3. We aimed to determine

Max Number of Active Agents	Percent Improvements	
	# col	S%
2	92.4%	83.1%
3	74.7%	64.2%
5	39.7%	69.4%

Table 3: Comparison of our model against a simplified variant that has no graph transformer communication protocol for $n = 10$ agents.

whether there was a relationship between an agent’s active status and performance. Specifically, we were interested in determining whether the performance of our algorithm could be attributed to large numbers of agents all active at the same time. To that end, we performed a modified experiment where we fixed the maximum number of agents that could be active at the same time. Then, we compared the difference in performance with and without our graph transformer protocol.

We note that across all numbers of active agents evaluated, the graph transformer communication protocol led to improvements in goal-reaching and lowered collision rates. The largest improvements from the graph transformer communication protocol occur at smaller numbers of active agents (e.g., 2). These results suggest that in dynamic graphs, smaller graph structures lead to reduced noise and a stronger abstraction of the essential structure of the environment.

5.5.2 Reward Formulation

In developing AsynCoMARL, we experimented with several different reward structures to balance trade-offs between individual goal-reaching and collaborative path planning. We compare the following reward formulations.

- **Repeated Reward:** This is the reward structure used in the InforMARL algorithm [13]. For a single agent, the reward is calculated by Equation 3.

In this formulation, the goal-reaching reward, $\mathcal{R}_{goal,t}^{(i)} = 5$, is given to each agent for every step it is at the goal. This means that even when an agent has successfully reached its goal and no longer needs to take any further control actions, it is still receiving a reward. This individual agent reward is combined with the reward received by all other agents to calculate the total reward for that step, $\mathcal{R}_{total}^{(i)}$. We combine the individual rewards of each agent into a sum to encourage collaborative behavior. With this reward structure, the largest possible reward is when all n agents reach their goal.

- **Piecewise Reward:** In the piecewise structure, each agent i receives a $\mathcal{R}_{goal,t}^{(i)}$ value of +5 at the first time step that it reaches gets within distance δ of the goal, p_i^{goal} . For every time step after, the goal-reaching value is changed in Equation 3 to be a smaller value.

$$\mathcal{R}_{goal,t}^{(i)} = \begin{cases} +5 & t_g = \|s_t^{(i)} - p_i^{goal}\| \leq \delta \\ +0.5 & t > t_g \end{cases} \quad (7)$$

In our analysis, we found that a larger magnitude of goal-reaching reward could obscure penalties for goal-reaching and collision avoidance for the other agents. By adopting the goal-reaching reward to be smaller after the first instance, the objective of the piecewise reward function is to encourage goal-reaching behavior without obscuring the collision and goal-reaching penalties of the other agents. We relied on the same summation function across all agents to encourage collaboration.

- **Single Goal-Reaching Reward:** In the single goal-reaching reward structure, each agent receives a $\mathcal{R}_{goal,t}^{(i)}$ value of +5 at the first time step that it reaches the goal. For every time step after, it receives no reward. We use boolean variable ϕ to designate that the goal-reaching reward has already been allocated to agent i . We investigated this structure to determine if there were any residual benefits to learning if the agents received no rewards after they reached their goals. As in the continuous reward structure, we relied on the same summation function across all agents to encourage collaboration.
- **Single Goal-Reaching Reward + Active Agent Sharing:** We adapt the single goal-reaching reward structure to also consider a more complex sharing function amongst agents. In this reward structure, agents only receive collaborative rewards if they communicate with one another during that timestep t . The purpose of this reward structure was to investigate if there was any impact on learned behaviors when agents received information more recently (and when they could operate synchronously for that given step).

Reward Structure	Metrics		
	T	# col	S%
Repeated	0.33	4.7	16%
Piecewise	0.23	1.6	45%
Single	0.31	1.21	41%
Single + Active	0.24	0.45	97%

Table 4: Comparison of the impact of several reward formulations on the resultant performance in the asynchronous setting for $n = 3$ agents.

Table 4 compares the performance of the different reward structures. Across the four reward structures, we have found that the *single goal-reaching reward + active agent sharing* case produced the best results in terms of success rate and average collision number.

This result indicates that at an individual level, learning improves in an asynchronous setting when the goal-reaching reward is only received once (as demonstrated by the lower collision rates for the two single-goal-reaching reward columns). By removing the repeated additive goal-reaching reward, the other agents are able to better refine their behaviors and recognize collisions.

When comparing the single-goal reaching reward and the active-agent sharing case, we have empirically found that the extent of shared collaboration plays an important role in the success rate of the agents. When the agents had a

shared reward, this created a lagged reward function, where the reward structure was determined by the last reward produced by one of the agents. By comparison, when active agents share their reward, this creates a reward function that is informed by the rewards produced by the actions taken at that step.

6 Conclusion

We introduced AsynCoMARL, a graph-transformer communication protocol for asynchronous multi-agent reinforcement learning designed to address the problem of coordination in environments where agents cannot communicate regularly. Each agent’s graph transformer utilizes a dynamic, weighted, directed graph to learn a communication protocol with other active agents in its vicinity. First, we showed that our method required less communication between agents and still produced similar success and collision rates as other multi-agent reinforcement learning approaches. Then, we evaluated AsynCoMARL’s performance in the more challenging Rover-Tower environment and found that our framework produces comparable results to other methods that require a separate network for the two agent classes. We further examined the workings of our graph transformer mechanism over the course of an episode and found that it effectively balances the trade-offs between the proximity of other agents and their active status.

Through ablation studies, we demonstrated the effectiveness of our graph transformer-based communication protocol, as well as the importance of reward structures in asynchronous settings. In future research, we aim to explore more advanced communication protocol architectures that can model different action-communication constraints common in real-world settings. Additionally, we want to investigate the feasibility of integrating additional mechanisms like control barrier functions to reduce the overall number of collisions.

7 Acknowledgements

The authors would like to thank the MIT SuperCloud [36] and the Lincoln Laboratory Supercomputing Center for providing high-performance computing resources that have contributed to the research results reported in this paper. This work was supported in part by NASA under grant #80NSSC23M0220 and the University Leadership Initiative (grant #80NSSC20M0163), but this article solely reflects the opinions and conclusions of its authors and not any NASA entity. The research was sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notion herein. Sydney Dolan was supported in part by the National Science Foundation Graduate Research Fellowship under Grant No. 1650114. J. Aloor was also supported in part by a Mathworks Fellowship.

References

- [1] Issa A.D. Nesnas, Lorraine M. Fesq, and Richard A. Volpe. Autonomy for space robots: Past, present, and future. *Current Robotics Reports*, 2(3):251–263, Jun 2021. doi:[10.1007/s43154-021-00057-2](https://doi.org/10.1007/s43154-021-00057-2). pages 1
- [2] Antoni Martorell-Torres, José Guerrero-Sastre, and Gabriel Oliver-Codina. Coordination of marine multi robot systems with communication constraints. *Applied Ocean Research*, 142:103848, 2024. ISSN 0141-1187. doi:<https://doi.org/10.1016/j.apor.2023.103848>. URL <https://www.sciencedirect.com/science/article/pii/S0141118723003899>. pages 1
- [3] W. Burgard, M. Moors, C. Stachniss, and F.E. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386, 2005. doi:[10.1109/TRO.2004.839232](https://doi.org/10.1109/TRO.2004.839232). pages 1
- [4] Sydney Dolan, Siddharth Nayak, and Hamsa Balakrishnan. Satellite navigation and coordination with limited information sharing, 2023. pages 2, 7
- [5] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning, 2019. URL <https://arxiv.org/abs/1810.02912>. pages 2, 7, 8, 9
- [6] Changxi Zhu, Mehdi Dastani, and Shihan Wang. A survey of multi-agent reinforcement learning with communication, 2022. URL <https://arxiv.org/abs/2203.08975>. pages 2
- [7] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation, 2016. URL <https://arxiv.org/abs/1605.07736>. pages 2
- [8] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks, 2018. URL <https://arxiv.org/abs/1812.09755>. pages 2

- [9] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *CoRR*, abs/1805.07733, 2018. URL <http://arxiv.org/abs/1805.07733>. pages 2
- [10] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Michael G. Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. *CoRR*, abs/1810.11187, 2018. URL <http://arxiv.org/abs/1810.11187>. pages 2
- [11] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *ICLR*, 2020. pages 2, 8
- [12] Sheng Li, Jayesh K. Gupta, Peter Morales, Ross Allen, and Mykel J. Kochenderfer. Deep implicit coordination graphs for multi-agent reinforcement learning, 2021. URL <https://arxiv.org/abs/2006.11438>. pages 2
- [13] Siddharth Nayak, Kenneth Choi, Wenqi Ding, Sydney Dolan, Karthik Gopalakrishnan, and Hamsa Balakrishnan. Scalable multi-agent reinforcement learning through intelligent information aggregation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 25817–25833. PMLR, 23–29 Jul 2023. URL <https://proceedings.mlr.press/v202/nayak23a.html>. pages 2, 11
- [14] Yaru Niu, Rohan R Paleja, and Matthew C Gombolay. Multi-agent graph-attention communication and teaming. In *AAMAS*, volume 21, page 20th, 2021. pages 2
- [15] Akshat Agarwal, Sumit Kumar, and Katia Sycara. Learning transferable cooperative behavior in multi-agent teams, 2019. pages 2, 4
- [16] Cameron Nowzari, Eloy Garcia, and Jorge Cortés. Event-triggered communication and control of networked systems for multi-agent consensus. *Automatica*, 105:1–27, 2019. ISSN 0005-1098. doi:<https://doi.org/10.1016/j.automatica.2019.03.009>. URL <https://www.sciencedirect.com/science/article/pii/S000510981930130X>. pages 2
- [17] Guangzheng Hu, Yuanheng Zhu, Dongbin Zhao, Mengchen Zhao, and Jianye Hao. Event-triggered multi-agent reinforcement learning with communication under limited-bandwidth constraint, 2020. URL <https://arxiv.org/abs/2010.04978>. pages 2
- [18] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Efficient communication in multi-agent reinforcement learning via variance based control, 2019. URL <https://arxiv.org/abs/1909.02682>. pages 2
- [19] Shuai Han, Mehdi Dastani, and Shihhan Wang. Model-based sparse communication in multi-agent reinforcement learning. *AAMAS '23: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, page 439–447, May 2023. pages 2
- [20] Kunal Menda, Yi-Chun Chen, Justin Grana, James W. Bono, Brendan D. Tracey, Mykel J. Kochenderfer, and David Wolpert. Deep reinforcement learning for event-driven multi-agent decision processes. *IEEE Transactions on Intelligent Transportation Systems*, 20(4):1259–1268, April 2019. ISSN 1558-0016. doi:[10.1109/tits.2018.2848264](https://dx.doi.org/10.1109/TITS.2018.2848264). URL <http://dx.doi.org/10.1109/TITS.2018.2848264>. pages 2
- [21] Christopher Amato, George D. Konidaris, and Leslie Kaelbling. Planning with macro-actions in decentralized pomdps. *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2012. pages 3
- [22] Yuchen Xiao, Weihao Tan, and Christopher Amato. Asynchronous actor-critic for multi-agent reinforcement learning, 2022. URL <https://arxiv.org/abs/2209.10113>. pages 3
- [23] Sunghoon Hong, Whiyoung Jung, Deunsol Yoon, Kanghoon Lee, and Woohyung Lim. Agent-oriented centralized critic for asynchronous multi-agent reinforcement learning. In *The Sixteenth Workshop on Adaptive and Learning Agents*, 2024. URL <https://openreview.net/forum?id=qfAY7DoJaD>. pages 3
- [24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017. URL <http://arxiv.org/abs/1706.02275>. pages 4
- [25] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of PPO in cooperative multi-agent games. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL <https://openreview.net/forum?id=YVXaxB6L2P1>. pages 4, 8
- [26] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *CoRR*, abs/2009.03509, 2020. URL <https://arxiv.org/abs/2009.03509>. pages 5

- [27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL <http://arxiv.org/abs/1707.06347>. pages 6
- [28] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL <https://arxiv.org/abs/1506.02438>. pages 6
- [29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL <https://arxiv.org/abs/1412.6980>. pages 6
- [30] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020. pages 7
- [31] David A. Vallado and Wayne D. McClain. *Fundamentals of astrodynamics and applications*. Microcosm Press, 2007. pages 7, 15
- [32] Matteo Gallici, Mario Martin, and Ivan Masmitja. Transfqlmix: Transformers for leveraging the graph structure of multi-agent reinforcement learning problems. *arXiv preprint arXiv:2301.05334*, 2023. pages 7, 8, 9
- [33] Jingqing Ruan, Yali Du, Xuantang Xiong, Dengpeng Xing, Xiyun Li, Linghui Meng, Haifeng Zhang, Jun Wang, and Bo Xu. Gcs: Graph-based coordination strategy for multi-agent reinforcement learning, 2022. URL <https://arxiv.org/abs/2201.06257>. pages 8, 16
- [34] Chao Yu, Xinyi Yang, Jiaxuan Gao, Jiayu Chen, Yunfei Li, Jijia Liu, Yunfei Xiang, Ruixin Huang, Huazhong Yang, Yi Wu, and Yu Wang. Asynchronous multi-agent reinforcement learning for efficient real-time multi-robot cooperative exploration, 2023. URL <https://arxiv.org/abs/2301.03398>. pages 8, 9
- [35] Xinran Li and Jun Zhang. Context-aware communication for multi-agent reinforcement learning, 2024. URL <https://arxiv.org/abs/2312.15600>. pages 8
- [36] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018. pages 12
- [37] Jennifer A Roberts and Peter C. E. Roberts. The development of high fidelity linearised j2 models for satellite formation flying control. In *14th AAS/AIAA Space Flight Mechanics Meeting*, Feb 2004. pages 15
- [38] Ulrich Walter. Orbit perturbations. *Astronautics*, page 555–660, 2018. doi:[10.1007/978-3-319-74373-8_12](https://doi.org/10.1007/978-3-319-74373-8_12). pages 15

8 Appendix

8.1 Baseline Implementation Details

We rely on the following implementations for each baseline and provide links to those implementations here. Note that we used the same hyperparameters as used in their original implementations, assuming that they were optimal.

1. asyncMAPPO: https://github.com/yang_xy20/async_mappo/tree/main
2. GCS: https://github.com/LXXXXR/GCS_aamas337/tree/master
3. DGN: https://github.com/jiechuanjiang/pytorch_DGN
4. CACOM: <https://github.com/LXXXXR/CACOM/tree/main>
5. Actor-Attention Critic: <https://github.com/shariqbal2810/MAAC/tree/master>
6. TransfQmix: https://github.com/mttga/pymarl_transformers/tree/main

8.2 Environment Implementation Details

We rely on the following implementations for the two environments we used in our experiments.

1. Cooperative Navigation: <https://github.com/sydneyid/satellite-cooperative-nav>
2. Rover-Tower: <https://github.com/shariqbal2810/MAAC/tree/master>

8.3 Cooperative Navigation Environment Description

There are n agents and n goals, along with static obstacles in the environment. Each agent is supposed to go to its distinct goal while avoiding collisions with other entities in the environment. Agents start at random locations at the beginning of each episode; the corresponding goals are also randomly distributed. Agents are governed by Clohessy-Wiltshire Equations [31]:

$$\ddot{x} - 3n^2x - 2n\dot{y} = u_x \quad (8)$$

$$\ddot{y} + 2n\dot{x} = u_y \quad (9)$$

$$\ddot{z} + n^2z = u_z \quad (10)$$

The above equations consider a localized coordinate system centered around one of the satellites, referred to as the *target*). The target satellite is assumed to have a circular orbit with an orbital rate of ω_n . The coordinates are defined such that x is measured radially outward from the target, y is along the orbit track of the target body, and z is along the angular momentum. u_x , u_y , and u_z represent the acceleration in the x,y, and z- directions. A full derivation of the perturbed equations can be found in [37]. While there exist additional environmental perturbations, such as solar radiative pressure or three-body effects, their impacts are magnitudes smaller [38]. In our method, we only use the x and y to dictate the agent's motion, as we are assuming all agents act within the same z plane. The experimental setup for each result is reported in Table 5.

Num Agents	Num Obstacles	World Size	Episode Length
3	3	2	125
5	3	2	125
7	3	3	125
10	3	3	125

Table 5: Experimental set up for experiments in Results section

8.4 Rover-Tower Environment Description

In the Rover-Tower environment, there are 4 rovers and 4 towers. Tower agents can send one of 5 discrete communication messages to their paired rover at each time step. Towers cannot move but can communicate with rovers so that rovers can move towards their corresponding goal. Each pair of rover and tower are negatively rewarded by the distance of the rover to its goal at each episode. In our setup, the communication is integrated into the environment (in the tower’s action space and the rovers observation space), rather than being explicitly part of the model.

8.5 GCS Extended Training Result

Algorithm	Broad Cast	Message Encoding	N=10		
			T	# col	S%
GCS [33]	Global	GAT	0.39	5.97	100

Table 6: Results for GCS when the training period is extended to 5,000,000 steps. Evaluations across 100 episodes with $n = 10$ agents.

8.6 Associated Key Dependencies

We rely on the following package versions to support our algorithm:

1. gym = 0.26.2
2. torch = 1.13.1
3. torch-geometric = 2.3.1
4. tensorboardX = 2.6.2.2
5. wandb = 0.17.4

8.7 Hyperparameters

Common Hyperparameters	Value
number of att heads	3
GAT Encoder num heads	4
num layers	4
decoder hidden dim	64

Table 7: Common Hyperparameters used in GCS

Common Hyperparameters	Value
recurrent data chunk length	10
gradient clip norm	10.0
gae lambda	0.95
gamma	0.99
value loss	Huber loss
huber delta	10.0
batch size	num envs \times buffer length \times num agents
mini batch size	batch size / mini-batch
optimizer	Adam
optimizer epsilon	1e-5
weight decay	0
network initialisation	Orthogonal
use reward normalisation	True
use feature normalisation	True
num envs	64
buffer length	125

Table 8: Common Hyperparameters used in asyncMAPPO, GCS, and AsynCoMARL

Common Hyperparameters	Value
attention dim	32
hidden layer size	64
mi loss weight	0.001
entropy loss weight	0.01
encoder dimension	8
request dimension	10
response dimension	20
3 Agent obs segments	$\langle 1 \times 4, 4 \times 2, 1 \times 6 \rangle$

Table 9: Common Hyperparameters used in CACOM