Leveraging Information Sharing for Satellite Navigation and Coordination

by

Sydney Dolan

B.S., Purdue University (2018)
S.M., Massachusetts Institute of Technology (2021)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2025

Authored by: Sydney Dolan
Department of Aeronautics and Astronautics
September 27, 2024

Certified by: Hamsa Balakrishnan
William E. Leonhard (1940) Professor of Aeronautics and Astronautics
Thesis Supervisor

Richard Linares
Rockwell International Career Development Professor of Aeronautics and Astronautics
Committee Member

Behçet Açikmeşe
Professor of Aerospace Engineering, University of Washington
Committee Member

Accepted by: Jonathan P. How
Richard Cockburn Maclaurin Professor of Aeronautics and Astronautics
Chair, Graduate Committee

Leveraging Information Sharing for Satellite Navigation and Coordination

by

Sydney Dolan

Submitted to the Department of Aeronautics and Astronautics
on September 27, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN AERONAUTICS AND ASTRONAUTICS

ABSTRACT

As the number of objects in orbit grows, so does the risk of collisions. The sheer volume of collision warning messages far exceeds the capacity of human analysts, placing a significant burden on satellite operators and underscoring the need for autonomous, decentralized traffic management. Unlike centralized conjunction analysis, decentralized space traffic management distributes coordination across multiple independent nodes, allowing satellites to collaborate directly. This approach could enhance the resilience, speed, and international cooperation of space operations, helping to manage the space environment.

For decentralized space traffic management to be viable, satellites must possess an accurate understanding of both the locations and intentions of other satellites. While satellites have precise knowledge of their own state, this accuracy diminishes when predicting the state of others. This gap is due to the limitations of onboard measurement systems and knowledge of each satellite's structure, configuration, and maneuverability. Such differences motivate the exploration of information sharing between operators to improve coordination. Sharing information could benefit both individual operators and the broader space community by enabling more accurate trajectory predictions, facilitating formal maneuver negotiations, and enhancing overall orbital safety and efficiency.

The main contribution of this thesis is to develop methods for autonomous satellite decision-making. By advancing the state of satellite autonomy, we can enhance high-level decision-making processes, enabling more adaptive and intelligent satellite coordination. This thesis begins by developing a multi-agent reinforcement learning environment to simulate satellite interactions in complex, high-dimensional settings. Then, we relax the assumption on synchronous communications and explore an alternate learning framework that relies on asynchronous communication between satellites. Our final contribution lies in a game-theoretic model of operator behavior in non-cooperative settings. Space is a competitive environment, and willingness to collaborate is mixed. As a result, we use game theory to obtain strategies to determine maneuvering and timing.

Thesis supervisor: Hamsa Balakrishnan
Title: William E. Leonhard (1940) Professor of Aeronautics and Astronautics

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Space presents many unique challenges that underscore the need for advanced robotics and artificial intelligence. The inherent dangers of space, like radiation, extreme temperatures, and microgravity, make it a hostile environment for human exploration, thus reinforcing the need for robotic systems. As the number of satellites and associated sensing capabilities continues to grow, the resultant amount of data generated far exceeds what human analysts can process. Advancement in satellite autonomy holds the power to address these issues; developing more capable satellites that can independently make decisions on their own can lead to advanced capabilities like conflict negotiation and data curation.

One of the pressing needs for satellite autonomy is in space traffic management. Until recently, the annual increase of objects in orbit remained around 300 per year, but factors like increased launch cadence and the emergence of large mega-constellations have led to a marked increase in the overall growth of the space population. The highest spatial density of objects is in the Low Earth Orbit (LEO) environment, a region of space around Earth between altitudes of 160 km and 2,000 km. This increased density has reached a point where satellite interactions are now frequent. Most simulations and forecasts indicate that the density of objects in LEO will continue to rise because the rate of new launches, on-orbit explosions, and collisions exceeds the LEO environment's natural ability to clean itself through orbital decay [3]. The increasing accumulation of operational payloads and uncontrollable debris presents a rising collision threat to active satellites, potentially disrupting essential services like communication, weather forecasting, navigation, scientific observation, and human spaceflight missions.

The consequences of even a single collision are significant. In 2009, the first hypervelocity collision occurred between Iridium 33 and Cosmos 2251, resulting in more than 1800 pieces of trackable debris. This debris creation represented roughly 10% of the tracked satellite population at the time, and resulted in an estimated 200,000 untrackable pieces of debris sized 10 centimeters or smaller[4]. While the Iridium-Cosmos collision was the first hypervelocity collision event, more collision events have occurred, like Cosmos 2535 in January 2020, Yunhai1-02 collision in March 2021, and Cosmos 1408 in November 2021 [5, 6, 7]. Though not classified as formal collision events, miscommunications between active operators are common and have led to numerous near-misses [8, 9].

Due to overall concern about collision risk, there have been renewed efforts to establish regulations and measures for space traffic management. These efforts aim to improve space

situational awareness and support collision avoidance and coordination among satellite operators. Organizations such as the United States Space Force 19^{th} Space Defense Squadron (19 SDS), the European Space Agency and other international space agencies work towards cataloging and monitoring objects in space. They maintain extensive databases that contain information on known objects in orbit, including their orbital parameters and characteristics. These databases help track and predict the movement of objects in space. In the case of the 19 SDS, they perform a satellite conjunction analysis, which assesses collision risk during a close encounter between a satellite and another object in orbit. This service is done for all commercial, civil, military, and academic operators. In the event that the conjunction assessment passes a certain threshold for collision risk, the 19 SDS issues a conjunction data message to the affected operators. Conjunction data messages (CDMs) are standardized data messages that describe spacecraft conjunction information, including miss distance, probability of collision, time of closest approach, and closest approach relative position and velocity [10]. Due to the vast number of potential interactions within the space population, the resultant overwhelming volume of conjunction data messages exceeds the capacity of human analysts to process and respond effectively. In 2023, the 19 SDS generated approximately 600,000 conjunction data messages per day, a 200% increase from 2020 [11]. If we assume that a small fraction of these are high probability events that could lead to an eventual collision, the scale of messages being delivered places a high burden on satellite operators seeking to mitigate them.

The burdening of manual collision avoidance services motivates the development of autonomous decentralized space traffic management. In contrast to traditional centralized conjunction analysis performed by the 19 SDS, decentralized traffic management refers to a system where the management and coordination of satellite movements are distributed across multiple independent nodes and entities. In such a system, satellites and satellite operators communicate and collaborate directly with one another to ensure safe navigation and avoid conflicts. Decentralized space traffic management services could enhance the resilience and responsiveness of space operations, enabling faster decision-making, reducing single points of failure, and fostering greater collaboration among international stakeholders in a congested space environment.

1.1 Benefits of Decentralization

Decentralized approaches to space traffic management have benefits beyond formal maneuver negotiation in the event of a potential collision. Each satellite has the most knowledge about its overall state. This knowledge includes information about the satellite's exact position, velocity, and attitude. The satellite also has information about its physical sizing, such as its hard body radius, which is used to inform realistic ballistic coefficients and probability of collision estimates. The satellite also possesses accurate data on its onboard fuel levels and thruster maneuvering capabilities. It also has the most reliable information on its intentions, including scheduled maneuvers, planned ephemerides, and current ephemerides. This understanding gives individual operators a far more accurate understanding of their satellites than what can be obtained from centralized space traffic management operations like 19 SDS or commercial space situational awareness services. If this individual information were

accessible to others in orbit, satellite operators could accurately propagate future satellite trajectories to confirm or deny collision warnings. Additionally, improved space situational awareness could help satellite operators assess any future secondary collisions that could occur as a result of their initial collision avoidance maneuvers.

However, operators are conflicted about what information to share, how often to share, and with whom, if at all. There is concern from legacy operators that providing too much insight into satellite's abilities and exact movements will cause satellite operators to lose their competitive edge, as competitors will be able to model their current capabilities accurately [12]. On the other hand, there is also concern that certain operators may not have quality ephemerides to share and thus will not offer anything of value when compared against more capable satellites [13]. Huge constellation operators such as SpaceX, Iridium, SES, and OneWeb [14, 15, 16, 17] all supported maneuverability information sharing for conjunction mitigation. In contrast, operators that do not maintain constellations, both new space and legacy operators, opposed information sharing due to concerns that specific information on satellite maneuverability is proprietary [18, 13]. From the perspective of military payloads, information sharing has the potential to jeopardize national security and place military space assets at risk.

There is a need for research on decentralized space traffic management to explore ways to incentivize the value and adoption of these types of systems. While it is intuitively expected that decentralized coordination among sets of operators could lead to improvements in collision avoidance, there have yet to be developmental frameworks in this area. Space traffic management modeling frameworks are difficult to develop due to the complexity of both the environmental dynamics and the diverse nature of the actors involved. In this thesis, we apply multi-agent reinforcement learning to this problem, allowing us to model multiple interacting agents making decisions in a shared environment.

1.2 Why Use Reinforcement Learning?

Against this backdrop, it is clear that there is a need to develop methodologies capable of addressing decentralized coordination between satellite operators in LEO. Approaches to modeling multiple spacecraft have ranged from optimal control to mixed-integer linear programming. However, these approaches quickly become computationally infeasible due to the number of constraints associated with each individual spacecraft. While it is possible for satellites to have a planned set of maneuvers encoded in advance to avoid this scaling problem, satellites will also need the ability to adapt as new collision avoidance scenarios arise. Moreover, such hard-coded maneuvers may become insufficient in an evolving space environment.

Reinforcement learning (RL) is a learning framework centered on the sequential interaction between a decision-making entity, known as the agent, and its environment. The agent observes the current state of the environment and selects an action accordingly. As a result of the action, the environment transitions to a new state, continuing the sequence. The agent's objective is to learn the optimal action for each state, which is captured in its policy. The agent learns by receiving rewards after each action, which assesses the quality of the action in that state. Over time, the agent refines its policy using this feedback.

Multi-agent reinforcement Learning (MARL) is an extension of reinforcement learning (RL) where multiple agents interact with a shared environment, each with their own goals and strategies. In MARL, these agents learn simultaneously, often in a dynamic and complex setting where one agent's actions can impact others' outcomes. MARL is particularly well-suited for problems like collaborative robotics, autonomous vehicle fleets, distributed resource management, and strategic games, where the interactions and dependencies among multiple agents are central to the problem's dynamics. Its strengths are best demonstrated in problems with the following characteristics:

1. **Supervision is costly or not possible**. In scenarios where human supervision is expensive or infeasible, MARL provides a means to learn policies by embedding system-level objectives into reward functions and utilizing exploration during training. This approach is especially relevant for NP-hard combinatorial optimization problems, where MARL can generate approximate solutions.

2. **Dynamic Settings with Changing Conditions**. MARL adapts to dynamic settings by continuously learning from real-time feedback, exploring new strategies, and updating policies based on interactions with a changing environment and other agents through the training process. This learned flexibility is useful in dynamic settings where conditions frequently change.

3. **Multi-agent Interaction**. MARL is effective for modeling scenarios where agents adapt their behaviors based on the environment and the actions of other agents. By distributing the learning and decision-making processes, MARL handles complex problems that would be impractical for a single-agent approach.

These characteristics are present in autonomous, decentralized space traffic management.

1.3 Contributions of This Thesis

The dissertation begins by detailing the related literature that our work is situated within or builds on (Chapter 2). This is followed by a series of chapters that detail our work exploring decentralized space traffic management and understanding how decentralized space traffic management strategies work in cooperative and uncooperative settings.

A Decentralized MARL Approach Towards Information Sharing: This chapter presents our decentralized multi-agent reinforcement learning model for space applications. First, we discuss the simulation environment that we created for proximity operations between satellites to train the MARL algorithm. We discuss the design of our new multi-agent reinforcement learning algorithm in this setting, which we've named SpaceMARL. Then, we present an initial analysis of the impact of decentralized coordination with local communication between satellites and compare the performance against centralized algorithms with perfect information sharing. Our empirical results show that our algorithm achieves near-optimal performance when compared to centralized methods despite having access to significantly less information. We also demonstrate how transfer learning can be used to accelerate training in this setting by initially training for a short period on a ground-based

model and using the resultant weights on the space environment. We show the generalizability of our algorithm, both towards different tasks and to perturbations in the environment, and comment on the consistent goal-reaching behaviors across these varied environments and navigation tasks. We use our graph formulation to determine the significance of sharing the intended goal location and find that goal sharing leads to improvements in path planning. These results provide initial insight into the underlying foundations behind learning-based controllers for collision avoidance in space.

Asynchronous and Communication Limited MARL: In this chapter, we relax the assumption that all communications and actions between agents must happen synchronously. We present AsyncCoMARL, a multi-agent reinforcement learning framework for coordination in asynchronous environments where actions and communications between agents happen at varying intervals. We find that graph transformers serve as effective encoders in this setting when compared to other graph-based attention mechanisms. We show the strength of independent actor-critic formulations in this setting and demonstrate that our algorithm possesses greater scalability than other asynchronous techniques.

Game Theoretic Models of Non-Cooperative Operator Interaction: This chapter studies collision avoidance strategies in non-cooperative settings. We formalized a single-step payoff matrix based on operator risk tolerance and maneuver cost and estimated the probability of collision. We evaluate the impact of the weighting on the risk tolerance and maneuver cost variables and find the emergence of a mixed Nash equilibrium strategy where both operators maneuver when there is a significantly higher cost to maneuver to operator risk ratio. We extend our work to the multi-step collision avoidance case, where satellite operators must decide when is best to maneuver during a set of n repeated games. We find that in our formulation for the repeated game setting, the optimal time to maneuver is earlier rather than closer to the time of the collision.

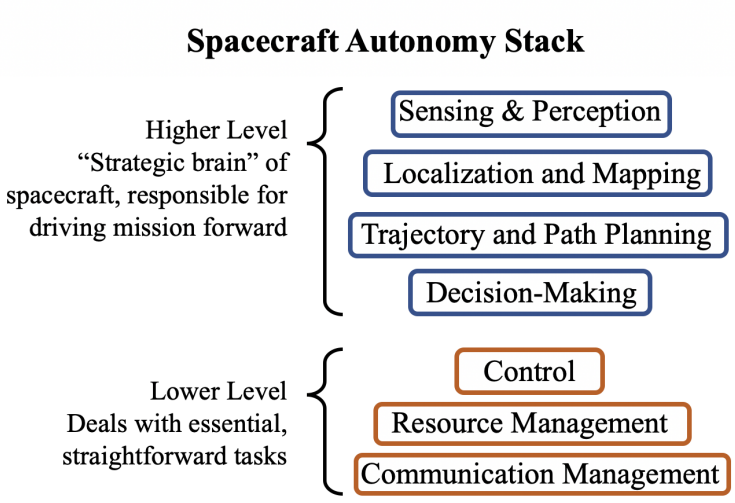


Figure 1.1: The main contribution of this thesis is to develop methods for autonomous satellite decision-making. By advancing the state of satellite autonomy, we can then address a need for higher levels in the autonomy stack for spacecraft.

Figure 1.1 contextualizes the contributions of this thesis in the overall spacecraft auton-

omy stack. The contributions of this thesis integrate into the spacecraft autonomy stack by enhancing high-level decision-making processes, enabling more adaptive and intelligent satellite coordination.

Finally, to conclude the thesis, in Chapter 6, we give an overview and conclusions about the work in this dissertation. We discuss the implications of this work for spacecraft autonomy as a whole and for space traffic management. Then, we present direct short-term and broader long-term possibilities for future work.

Chapter 2

Literature Review

First, we review reinforcement learning techniques for space applications. Then, we delve into the larger field of multi-agent reinforcement learning, highlighting relevant works on decentralized multi-agent coordination with limited communication. In the final section, we review advancements in game theory.

2.1 Reinforcement Learning for Space Applications

Reinforcement learning (RL) is seen as a promising technique for space applications due to its ability to learn control policies without an explicit mathematical model of the whole environment [19]. The algorithm learns how to adapt to a wide array of operations, system uncertainties, and off-nominal conditions. This generalizability is highly desirable for space missions that will feature complex and even previously unseen operational environments. While the training phase can be computationally expensive, the trained policies require limited memory and computational resources to run. As a result, RL has been successfully demonstrated in simulation for numerous space applications including spacecraft landing [20, 21, 22], trajectory design [23, 24, 25, 26], attitude control [27, 28, 29], docking [30, 31, 32], constellation orbital control [33, 34], autonomous inspection [35], and on-board decision making for spacecraft operations scheduling [36, 37, 38]. For a complete survey of existing reinforcement learning in spacecraft control applications, we refer the reader to [39].

Several works rely on relative motion equations to model the dynamic motion of satellites [31, 36, 40]. Federici et al. used RL to design a controlled guidance algorithm for a spacecraft in a terminal rendezvous maneuver through the Clohessy-Wiltshire equations. With the assumption that the satellite is within the approach corridor and the target is always in the visibility cone, the chaser reference trajectory must lead it from the initial condition to the prescribed final one. A velocity perturbation is added at each step within the dynamics to address perturbations and stochastic effects. The RL algorithm develops a policy containing locations and directions for satellite burn maneuvers to enable successful rendezvous [31]. Huang et al. use graph theory and deep reinforcement learning to solve the spacecraft operations scheduling problem. Graph theory is used to model clusters of satellites as nodes, with edges representing links between tasks for scheduling. This approach allows for a simplified scheduling problem, where multiple satellite observations can be combined into one clustered

observation if there are edge connections between two vertices. Then, deep reinforcement learning is used to determine the schedule [36]. Gaudet et al. rely on meta-RL to adapt to novel partially observable environments. Meta-RL refers to the process of "learning to learn" by selecting the best hyperparameters for a previously unseen environment. They test the effectiveness of this meta-learning by demonstrating the performance of the RL algorithm on several experiments, including a Mars landing with an engine failure, a Mars landing with a state estimation bias, and an asteroid landing with unknown dynamics [40]. This work demonstrates the value of refined tuning mechanisms for RL, which can indeed adapt to environments with unknown or highly variable dynamics.

To our knowledge, there are no other works that use multi-agent reinforcement learning for spacecraft. Rather than determining a single guidance trajectory or sensor tasking schedule for one ground station, our work considers multiple satellites and their interactions with one another. When multiple satellites act in a shared environment such as space, their interests and intentions may be misaligned. For example, in a competition setting, satellites may belong to competing communications operators, which will discourage them from making their movements known to nearby actors. In contrast, in a cooperative environment, such as a servicing agent rendezvousing with its target, the two agents may work together to ensure the servicer can reach its goal. Through multi-agent reinforcement learning (MARL), we can obtain a richer understanding of the social complexities of different operations in space.

Prior research has found that a direct implementation of a single-agent RL to several agents cannot converge to optimal solutions because the environment dynamically changes each step due to the agent's movements [41]. Multi-agent reinforcement learning differs because the agents aim to jointly optimize a reward together, and the movement between agents is strategic. However, MARL has yet to be applied in many applications due to issues with scalability. The addition of more agents means that the solution space exponentially increases [41]. This makes training MARL agents more difficult and creates issues with convergence.

2.2 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) approaches allow the investigation of a range of interactions (e.g., competitive or cooperative) between satellite operators. To the best of our knowledge, there has yet to be a MARL-based control solution implemented in the space domain, but MARL algorithms have been successfully applied to problems in adjacent domains like air traffic control. Ghosh et al. design an ensemble learning framework with an integrated dual-learning structure. Locally, a kernel-based RL encoding is used to approximate an agent's local information. Globally, a deep MARL model is used to capture large-scale interactions. This joint kernel-MARL model was evaluated on an air traffic simulator and successfully directed 1668 aircraft over a 24-hour period while achieving more reward than baseline approaches [42]. Brittain et al. adapt the proximal policy optimization algorithm [43] to incorporate an attention network, which is used to identify priority information in the aircraft's state environment [44]. While not in air traffic control, Zhang et al. designed a MARL algorithm that used graph neural networks to augment communication and control Lyapunov barrier functions to guarantee safety [45]. Similar to our methodol-

ogy, this work also applied an entity-graph structure, with nodes representing agents and obstacles and edges representing communication channels. This method was demonstrated successfully in simulation on ground-based robots and on drone hardware.

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

Through communication, agents can obtain a better understanding of their own environment and of other agents, thus improving their ability to coordinate their behaviors. For comprehensive surveys on research on the impact of communication in multi-agent collaboration, we refer the reader to [46, 47]. Seminal works such as CommNet [48] use 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 [49, 50] aimed to investigate methodologies that improve coordination by using a small subset of agents. For example, ATOC [50] uses a probabilistic gate mechanism to communicate with nearby agents in an observable field and a bi-LSTM to concatenate messages together. Incorporating attention mechanisms to encode communications between agents led to further advancements in multi-agent collaboration. TarMAC [51] uses an attention mechanism to generate encodings of the content of messages so that nearby agents can learn the message's significance via 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 [52], DICG [53], and EMP [54] formulate interactions between different agents as a graph. These methods rely on dynamic and temporal 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 presumes that all agents have unrestricted communication to coordinate synchronously.

2.4 Event-Triggered Communication

Event-triggered control (ETC) is a control strategy in which the system updates its control actions only when certain events or conditions are triggered, rather than continuously or at regular time intervals as in traditional time-triggered control systems [55]. By reducing the number of control updates, ETC can significantly decrease the workload on controllers, a beneficial attribute for systems with limited resources. ETC [56], VBC [57], and MBC [58] 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 lower communication budgets. Menda et al. frame the asynchronous decision-making problem where agents

choose actions when prompted by an event or some set of events occurring in the environment [59]. By relying on a continuous state-space representation and an event-driven simulator, the agents step from event to event, and lower-level time-step simulations are eliminated. This improves the algorithm's scalability but leaves the agents unable to capture low-level agent-agent interactions within an event duration.

2.5 Asynchronous Actor Critic

An alternative approach to asynchronous environments formulates the problem as a Macro-Action Decentralized Partially Observable Markov Decision Process (MacDec-POMDP), where agents can start and end macro actions at different time steps [60]. Xiao et al. propose IAICC, a multi-agent policy gradient methodology that allows agents to asynchronously learn high-level policies over pre-defined macro actions [61]. Hong et al. alter the IAICC framework to encode agent time history independently to avoid duplicate macro-observations from non-macro actionable agents in the centralized critic [62]. These works focus on learning algorithms for asynchronous $macro-actions$ that occur across multiple time steps. Comparatively, our work focuses on the micro level, looking for planning opportunities while agents achieve the same macro-level task.

2.6 Game Theoretic Methods for Space Domain

Game theory provides the theoretical underpinnings for understanding and analyzing multi-agent systems, while MARL offers practical algorithms for learning in these systems when explicit modeling of the environment and agents is infeasible. Game theory is well-suited for non-cooperative settings, as it provides strategic frameworks for analyzing and predicting the behavior of independent agents with conflicting interests. As a result, given the contested environment of space we use game theory to account for the mixed intentions of the operators within the domain.

Game theory has been applied successfully in the space domain across various applications, including mega-constellation design [63], collision avoidance [64, 65], and debris removal [66]. Qureshi et al. introduce a multi-agent game that simulates the dynamics of both LEO constellations and the economic interactions between constellation operators [63]. By combining gamification with modeling, they propose a framework where multiple players develop and manage their own LEO constellations. They validate this model using Monte Carlo policy evaluation. This game targets competitive scenarios where each agent must manage its economic assets based on its LEO constellation's specific characteristics, limiting the framework's applicability to broader space traffic management contexts. Klima et al. apply evolutionary game theory to examine the space debris removal challenge, focusing on whether agencies (e.g., NASA, ESA) should invest in the costly yet widely beneficial process of debris removal or rely on others to act first [66]. This dilemma creates a classic 'tragedy of the commons': if all agencies delay action, no debris is cleared, resulting in severe consequences for everyone. The study employs empirical game theory and simulations to evaluate various strategies in two- and three-player scenarios. Their analysis shows

that the cost-benefit ratio of debris removal significantly shapes the game's dynamics, with strategic equilibrium either promoting or deterring cooperation among agencies, depending on this ratio. This game assesses the space environment on a global scale and does not offer strategic insights into the actions of individual satellites, and thus cannot be applied to our problem. Palafox et al. focus on the problem of collision between a set of satellites operating in proximity to one another [64]. They propose a dynamic game-theoretic method for multi-robot collision avoidance that relies on a set of rotating hyperplane constraints to separate robots and obstacles. The proposed method learns these hyperplane parameters from expert trajectories collected by observing human operators. Then, the parameters are optimized to match the equilibrium trajectory to the expert's behavior. Palafox et al. validate their method through tests and show that the learned parameters generalize well to scenarios involving more robots and previously unseen initial conditions [64]. Although this methodology is designed for collision avoidance, it specifically targets proximity operations rather than satellite collision avoidance, which generally involves distances on the scale of kilometers rather than meters. Additionally, this approach simplifies the collision avoidance problem by assuming perfectly known operator positions, whereas satellite collision avoidance must account for uncertainties in each satellite's exact location.

Satellite collision avoidance presents a complex game-theoretic problem due to the need for multiple independent entities to make interdependent decisions under conditions of uncertainty. In low Earth orbit, where many satellites share overlapping paths, operators must decide how to adjust orbits to avoid potential collisions while balancing fuel consumption, collision uncertainty, and the unpredictable maneuvers of other satellites. Each operator acts as a "player" in this game, where actions to minimize collision risk may inadvertently affect others, introducing a strategic layer to the decision-making process. This interdependence creates a non-cooperative setting where each operator has incomplete information about the exact position and intentions of others, further complicated by communication and information-sharing constraints. Game theory provides frameworks, such as Nash equilibrium and cooperative games, to model and analyze these interactions, helping operators predict the likely actions of others and strategize accordingly. By modeling satellite collision avoidance in this way, game theory allows for more robust, anticipatory strategies that can improve overall space traffic safety. The core challenge of satellite collision avoidance is similar to several well-known, high-stakes negotiation problems in game theory, such as the Cuban Missile Crisis [67, 68], climate change [69], deadlock in autonomous highway driving [70], and public goods dilemmas [71]. Our distinctive contribution to this field is the integration of game-theoretic frameworks with a high-fidelity satellite simulation that captures satellite characteristics and operator trade-offs.

Chapter 3

Multi-Agent Reinforcement Learning Approach to Information Sharing

3.1 Introduction

In a crowded space environment, satellites need to rely on communication from others nearby to perform effective collision-free operations. In this setting, satellites have limited information about the states and intentions of their neighbors. Approaches to modeling the multiple spacecraft have ranged from reachability to mixed integer linear programming [72, 73]. However, these approaches all scale poorly due to a coupling between the number of spacecraft and the resultant number of collision avoidance constraints. While it is possible for satellites to have a planned set of maneuvers encoded in advance to avoid this scaling problem, satellites will also need the ability to adapt as new collision avoidance scenarios arise. Moreover, in a dynamic and evolving space environment, such hard-coded maneuvers may become insufficient.

Multi-agent reinforcement learning (MARL) offers a possible alternative, as its framework seeks to learn behaviors between satellites that result in collision-free navigation. Instead of formulating the problem with n agents and $O(n^2)$ constraints, MARL provides a generalizable approach, as it learns a set of desirable behaviors from both its exploration of the environment and the resultant interactions with other satellites. One of the main limitations of multi-agent reinforcement learning is that it requires exhaustive training to produce a desirable policy, a problem further amplified by satellite-to-satellite interactions. To address this, we investigate using transfer learning to adapt a general MARL model, previously trained on ground agents, to a space environment. We extend our recent work in a multi-agent reinforcement learning setting (InforMARL [74]), and propose a new space environment-based formulation, which we refer to as SpaceMARL. We demonstrate that transfer learning from the ground to the space environment is remarkably effective: it achieves better sample complexity and slightly higher rewards than when directly training a model in the space environment. This result is despite the two environments being quite different in terms of the underlying dynamics that govern them. We then consider a variant of the space environment that accounts for perturbations in gravitational disturbances and find that transfer learning is still effective.

After demonstrating the performance of our model against comparable algorithms, we

then use our framework to study the role that information-sharing plays in satellite operator decision-making. Our work is motivated by the premise that satellites have limited observations of their neighbors but possess the best knowledge of their own location. Through our framework, which utilizes graph theory to model communication, we seek to understand both *what* to share and with *whom* to share. To do this, we rely on graph neural networks as a proxy for communication between agents. The graph neural network learns the optimal message passing prioritization to aid the satellites in achieving the best policy [75]. By evaluating the message prioritization and contents of the inputs to the graph neural network, we can study what information is most useful for satellite coordination. Specifically, we use our model to assess the impact of information sharing by evaluating changes in collision rates and path planning.

Our contributions include the following:

- We develop a space environment simulator for multi-agent reinforcement learning.
- We extend our prior work on multi-agent reinforcement learning [74] and introduce SpaceMARL, an approach that leverages graph neural networks to enhance communication for space applications.
- We rely on transfer learning to speed up the training process for SpaceMARL model and demonstrate its effectiveness against other common MARL algorithms.
- We demonstrate the performance of our multi-agent reinforcement learning-based approach on several formation and rendezvous tasks.
- We use our method to quantify the impact of different pieces of information by evaluating changes in performance under different graph structures.

In this chapter, we will explore the development of our space-focused methodology for multi-agent reinforcement learning. This chapter is based on work that was presented in [76]. First, we provide a brief introduction to reinforcement learning. Then, we discuss our graph abstraction that the MARL model uses to simulate communication and characterize relationships. We then discuss our dynamics models, and provide the status of current results.

3.2 Multi-Agent Reinforcement Learning Overview

The goal of multi-agent reinforcement learning is for each agent to learn how to complete a task through repeated interactions with the environment and other agents. Multi-agent reinforcement learning can be described by a partially observable Markov decision process (DEC-POMDP) [77, 78, 79] with shared rewards. A DEC-POMDP is defined by $\langle N, \mathcal{S}, \mathcal{O}, \mathcal{A}, R, P, \gamma \rangle$. N is the number of agents, $s \in \mathcal{S} = \mathbb{R}^{N \times \mathbb{D}}$ is the state space of the environment, $\mathbb{D}$ is the dimension of the state, and $o^{(i)} = \mathcal{O}(s^{(i)}) \in \mathbb{R}^d$ is the local observation of agent i, where $d \leq \mathbb{D}$ is the observation dimension. $a^{(i)} \in \mathcal{A}$ is the action space for agent i and the joint action for all N agents is given by $\mathcal{A} = (a^{(1)}, ..., a^{(n)})$. Specifically, in our environment, $a^{(i)}$ is a one-hot vector whose size is equal to the number of possible actions.

In our environment, this correlates with a vector of actions $a^{(i)} = (+x, -x, +y, -y, 0)$. While this work does not consider more advanced fuel usage techniques, the action structure itself is generalizable, and our framework could accommodate other methods. For example, with an appropriate simulation environment, a one-hot encoding could be used to represent the usage of a method, such as Gonzalo et al.'s analytical collision avoidance maneuver formulation [80]. $R(S, \mathcal{A})$ is the joint reward function. We adopt a similar reward function as used in the multi-agent particle environment (MAPE) [81] where the joint reward function is defined as $R(\mathcal{S}, \mathcal{A}) = \sum_{i=1}^{N} Rew_t^{(i)}$, which encourages cooperation among all agents. Here $Rew_t^{(i)}$ is each agent's reward at time step t and depends on the scenario. $P(s'|s, \mathcal{A})$ is the transition probability from s to s' given joint action $\mathcal{A}$. $\gamma \in [0, 1)$ is the discount factor.

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: $\langle \mathcal{N}, \mathcal{S}, \mathcal{O}, \mathcal{A}, R, P, \gamma, \mathcal{G} \rangle$. $g^{(i)} = \mathcal{G}(s; i)$ represents the graph network formed by the entities of the environment with respect to agent i. Features of the graph structure are used in our actor-critic learning architecture [82].

In actor-critic architectures, the actor aims to optimize a policy, and the critic evaluates the action produced by the actor through the value function. The actor's policy $\pi_\theta^i \left(a^{(i)}|o^{(i)}, g^{(i)}\right)$ is parameterized by θ, and relies on a local observation $o^{(i)}$ and the graph neural network $g^{(i)}$ to optimizing the discounted joint accumulated reward:
$J(\theta) = \mathbb{E}_{\mathcal{A},\mathcal{S}} \left[\sum_t \gamma^t R\left(\mathcal{S}, \mathcal{A}\right)\right]$. The critic will inform the actor how effective the action selection was using a value function for the environment $Q_\phi^{\pi_\theta}(h, \mathcal{G}, a)$, where h is the history maintained by a recurrent neural network.

During training, the critic has access to the observations from every agent. However, during execution, there is no centralized critic available, and each satellite is operating using the learned weights, θ, from the actor network. Thus, our work is able to take advantage of centralized training to improve its learned performance and a decentralized execution structure to ensure that the algorithm can work in decentralized settings [83].

3.3 Graph Overview

We rely on a graph neural network (GNN) to synthesize the communication between agents. Graph neural networks are an optimizable transformation on all attributes in the graph (nodes, edges) that preserves graph symmetries [84]. The output of a graph neural network is an encoding that represents the relationality between nodes, this encoding is used as an input to both the actor and the critic in our actor-critic learning process.

Graph neural networks require that the problem be translated into a graph structure before the algorithm can learn. Every object in the environment (also referred to as an 'entity') is assumed to be either a satellite, an obstacle, or a goal/landmark. In our formulation, every entity in the environment is a node on the graph. Each node j on the graph $g^{(i)}$ has node features $x_j = [p_i^j, v_i^j, p_i^{\text{goal},j}, \texttt{entity\_type(j)}]$ where $p_i^j, v_i^j, p_i^{\text{goal},j}$ are the *relative* position, velocity, and position of the goal of the entity at node j with respect to satellite 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 [85].

There exists an edge $e \in \mathcal{E}$ between a satellite and an entity if they are within a sensing radius ρ of each other. We note that while some satellites contain laser cross-links that enable continuous communication, the vast majority of satellite communications include time delays and transmission losses. In this work, we aim to study the value of information, rather than the mechanism by which it is sent. As the specific sensor class is highly variable and dependent on specific ground architectures, we have left the incorporation of additional time delays as future work. Edges can be bidirectional or unidirectional. A unidirectional edge is equivalent to the satellite sensing a nearby entity's state, while a bidirectional edge is equivalent to a communication channel between satellites. Each edge e_{ij} has an associated edge feature given by the Euclidean distance between the entities i and j. This edge feature is calculated by taking the difference between entity i and entity j. An example graph representation is shown in the left-most pane of Figure 3.1.

Using these node and edge definitions, we then define a satellite-entity graph with respect to satellite i at each time-step t, as $g_t^{(i)} \in \mathcal{G} : (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ is an entity in the environment. The variable `entity_type(j)`$\in \{$`satellite`, `obstacle`, `goal`$\}$ determines the type of entity at node j. This structure is similar to the satellite-entity graph defined in [54], but without the assumption that all satellites have access to the positions of all entities at the beginning of each episode. Note that our formulation also supports cases where disconnected sub-graphs are formed due to the positioning of the entities in the environment.

Figure 3.1 shows the complete message-passing process used to learn the relative significance of different communications to node i. The process first begins with each node i gathering node x_i^j and edge e_{ij} information from other connected nodes. The information from these connected nodes are aggregated together to produce x_{embed}. Then, each node passes this information through a Unified Message Passing Model (UniMP) [86] to learn their importance. A UniMP is a variant of a graph transformer [87] where each layer update is defined as $x_i' = W_1 \cdot x_i + \sum_{j \in \mathbb{N}(i)} \alpha_{i,j} W_2 \cdot x_j$, where x_j 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. $\alpha_{i,j}$ are computed via multi-head dot product attention:

$$\alpha_{i,j} = \text{softmax}\left(\frac{(W_3 \cdot x_i)^T (W_4 \cdot x_j + W_5 \cdot e_{ij})}{\sqrt{\text{c}}}\right) \tag{3.1}$$

where e_{ij} are edge features for the edge connecting nodes i and j, and c is the output dimension for that specific layer. The attention mechanism within the Unified Message Passing Layer allows the nodes to selectively prioritize messages coming from their neighbors according to their importance.

As shown in Figure 3.1, we employ multiple Unified Message Passing Layers. With the addition of successive message-passing layers, each node embedding changes to reflect the information they have processed. As nodes repeat the gathering process with their connected nodes in the subsequent message-passing layer, they are implicitly receiving information from second-hop neighbors. In our architecture, we experimented with different numbers of message-passing layers but found that two layers were sufficient to propagate information between nodes that are higher-order neighbors with each other. While we rely on successive message-passing layers in the graph neural network, we only require that each node

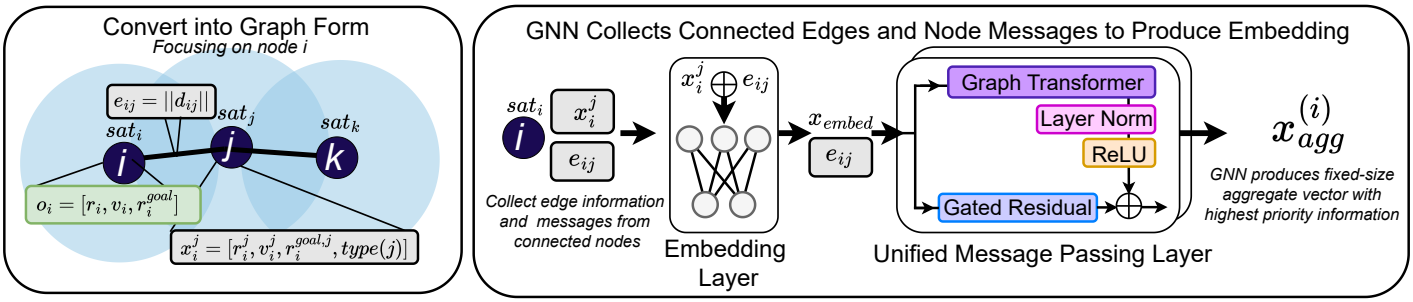


Figure 3.1: Graph Neural Network Architecture.

communicates once with its connected nodes for each step, dt.

The graph neural network produces fixed size vector $x_{\text{agg}}^{(i)}$, as shown by panel 2 in Figure 3.1. $x_{\text{agg}}^{(i)}$, is used as an input to the actor network, which relies on the embedding to determine the action for that specific time step. Because $x_{\text{agg}}^{(i)}$ is a fixed length embedding, this architecture allows SpaceMARL to dynamically adapt to a changing number of entities in the environment while remaining invariant to the permutation of the observed entities. Therefore, when the actor network is evaluated on more entities, it can still rely on the same training weights that it obtained when trained on fewer agents.

In training, the critic receives all the embeddings throughout the graph so that it has a global view of the environment. A global mean pooling operator, $X_{\text{agg}} = \frac{1}{N}\sum_{i=1}^{N} x_{\text{agg}}^{(i)}$, is applied to aggregate the updated node features in the graph. This aggregated node value X_{agg} is used as an input to the critic, which uses this information, in combination with the state-action pairs, to evaluate the action selection from the actor. Due to the pooling operator, training is transferable to a variable number of satellites, as the graph representation is fixed to size X_{agg} [88, 89].

We note that the actor and critic networks can be either a multi-layer perceptron (MLP) or a recurrent neural network (RNN) [90], using either LSTMs [91] or GRUs [92]. Our proposed information aggregation method can be used in conjunction with any standard MARL algorithm (eg, MADDPG [83], MATD3 [93], MAPPO [94], QMIX [95], VDN [96], etc.).

3.4 Transfer Learning

For several of our experiments, we rely on transfer learning to accelerate training. The main principle behind transfer learning is that experience gained in learning from performing one task can help improve performance in a related but different task [88]. In other applications, it has been shown to improve the results of existing algorithms and speed up the learning [97]. Transfer learning works by taking the relevant parts of a pre-trained RL framework and using them as an initialization for an entirely new environment. In this work, we train a model on the ground environment and save the resultant trained weights from the actor-critic network: the θ parameter from the policy gradient of the actor-network, and the value function, $Q_\phi^{\pi_\theta}$, from the critic network. Then, we use these parameters as an initialization

for training in the relative motion space model.

3.5 Environment

In this work, we consider two different environments: (1) A ground environment in which the agents' dynamics are governed by a double integrator physics model [98], and (2) a space environment in which the relative motion between two agents (satellites) follows the Clohessy-Wiltshire equations [99]. We also consider a modified space environment that accounts for perturbations in the dynamics caused by the oblateness of the Earth.

Ground environment: Double-integrator model

For the ground environment in our transfer learning experiments, we rely on a double-integrator physics model to simulate the motion of agents on the ground [98]. Our environment is a modification of the one used in [81]. It corresponds to a 2D space in which satellites move based on the following dynamics:

$$\ddot{x} + \frac{\gamma}{m}\dot{x} - \frac{f_x}{m}\tau = u_x \tag{3.2}$$

$$\ddot{y} + \frac{\gamma}{m}\dot{y} - \frac{f_y}{m}\tau = u_y \tag{3.3}$$

In the above equation, u_x and u_y are accelerations selected by the MARL algorithm in the $x-$ and $y-$ directions. f_x and f_y represent the x and y components of collision force on the agent. τ is used as a binary indicator to determine if the agent has collided with another agent. m is the mass of each entity, and γ is a damping coefficient. In our simulations of the ground environment, each agent starts from a random, stationary position. The environment size is a hyperparameter, defining the area available for an agent to move in.

Space environment: Clohessy-Wiltshire equations

In the space environment, the agents simulated are satellites. For two objects in orbit, we consider their *relative* motion to be governed by Clohessy-Wiltshire Equations [99]:

$$\ddot{x} - 3n^2x - 2n\dot{y} = u_x \tag{3.4}$$

$$\ddot{y} + 2n\dot{x} = u_y \tag{3.5}$$

$$\ddot{z} + n^2z = u_z \tag{3.6}$$

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. For the perturbed model, the dynamic equations are modified to include gravitational disturbances

(or perturbations) due to Earth's oblateness:

$$\ddot{x} - (5c^2 - 2)n^2 x - 2\omega_n c\dot{y} = u_x \tag{3.7}$$

$$\ddot{y} + 2ncx = u_y \tag{3.8}$$

$$\ddot{z} - (2 - 3c^2)n^2 z = u_z \tag{3.9}$$

In the above, c is a parameter that reflects the change in orbital rate experienced by the satellite due to these perturbations. The value of c is given by:

$$c = \sqrt{1+s}, \text{ where } s = \frac{3J_2 R_e^2}{8r^2}(1 + 3\cos(2i)). \tag{3.10}$$

In Equation 3.10, J_2 is a coefficient representing the gravitational effect of a body's oblateness, R_e represents the Earth's radius, r represents the radius of the path of the target satellite, and the inclination, i, is the angle of the orbit relative to Earth's equator. An angle of $i = 0°$ represents an equatorial orbit, whereas $i = 90°$ represents a polar orbit. We note that substituting $c = 1$ (i.e., $\phi = 54°$) in Equations 3.7-3.9 results in the Clohessy-Wiltshire equations (3.4-3.6).

A full derivation of the perturbed equations can be found in [100]. While there exist additional environmental perturbations, such as solar radiative pressure or three-body effects, their impacts are magnitudes smaller [101]. Consequently, these other perturbations only affect satellite dynamics over time periods that are much longer than the episode lengths considered in this paper, which are of the order of a few hours.

This work focuses on in-plane maneuvers, which means all satellites and debris involved are assumed to lie on the same xy orbital plane. Similar to the ground environment, the maneuvers are determined by the control action (u_x or u_y) recommended by the actor-critic algorithm.

In both the original Clohessy-Wiltshire equations and the perturbed model, z-dynamics is decoupled from those in the x- and y-directions. While not a topic of investigation in this paper, in principle, the same techniques could be applied to the cross-track dimension by training a separate model focused only on controlling the z-dynamics.

Comparison of ground and space environments

We explore the potential of transfer learning for this application because the space environment has complicated dynamics that are much more challenging for the algorithm to learn. From Equations 3.4 and 3.5, we see that the x and y variables in the Clohessy-Wiltshire equations are coupled. By contrast, the ground equations are independent of one another. Additionally, the dynamics in the space environment are more sensitive to actions selected by the agents [102].

In both cases, we adopt a simplistic reward function similar to the one used in multi-agent particle environment [81]. We assume that at time t, each agent i gets a reward: $Rew_t^{(i)} = Rew_{\text{dist},t}^{(i)} + Rew_{\text{coll},t}^{(i)} + Rew_{\text{goal},t}^{(i)}$, where $Rew_{\text{dist},t}^{(i)}$ is the negative of the Euclidean distance to the goal, $Rew_{\text{coll},t}^{(i)} = -5$ if it collides with any other entity and zero otherwise, and $Rew_{\text{goal},t}^{(i)} = +5$ if the agent has reached the goal and zero otherwise. The joint reward function

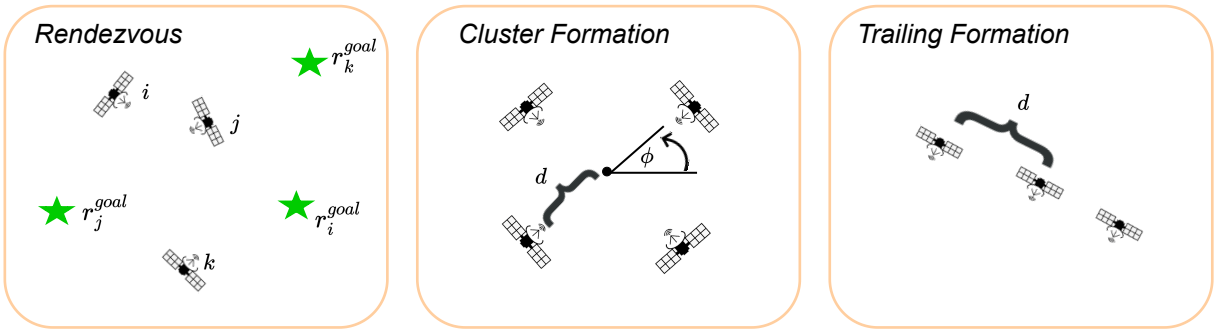


Figure 3.2: Visualization of the three control tasks.

is defined as the sum of the individual agent rewards, which encourages cooperation among all agents. It is worth noting that the reward function can be further refined, especially in terms of how collisions are penalized. Future research will include imposing larger penalties for collisions, as well as the use of control barrier functions for providing safety guarantees [103].

It is important to note that the size of the environment considered (4 sq km) along with the number of objects in it (between 6-13) result in unrealistically dense scenarios. The purpose of these experiments is to evaluate the efficiency of the methods and the general trends in performance, and not to determine values (e.g., of the collision rates) that are representative of real-world operations.

3.6 Experiments

In all these environments, N agents move around in a 2D space following the dynamics models outlined in Section 3.5 and 3.5. For the transfer learning model, the agents are first trained on a ground model, as mentioned in section 3.5. Subsequent training occurs in the space environment, but the actor-critic network is initialized using the weights that came from the ground environment training. For all other models, the agents are trained and evaluated using the space model mentioned in Section 3.5. The ground model used for transfer learning is trained for 200,000 steps. All other algorithms are trained for 2,000,000 steps. The learning parameters used for each algorithm can be found in Appendix A. Actions taken during training and evaluation are discrete and represent an instantaneous burn in the x- and y- directions in the CHW frame with thrust magnitude 1 N.

The sensing radius of the satellites is set to be 0.5 km. Satellites and obstacles are modeled as Lambertian spheres with a 1-meter radius. We determine that a collision occurs when there is an Euclidean difference between satellites smaller than 0.1 km. Satellites are initialized into an environment that is 2×2 km wide. The altitude of the circular orbit of the leader satellite is assumed to be 400 km, and the constants 6378 km and $\mu_e = 3.986 \times 10^5 km^3/s^2$ are used. Then, the radius of the circular orbit is $R_0 = 6778km$, the orbit rate is $n = 1.1291 \times 10^-3$rad/s.

There are several underlying assumptions common in all of our results. First, we are assuming that all satellites will be able to communicate with the others once per each step.

We assume that the communication between agents is synchronous and lossless, and can be accomplished within one step, dt. Similarly, we are assuming that the control action can be completed within one step, dt. Second, we are ignoring any fuel constraints associated with the thruster. Third, we are assuming that each satellite has perfect observations of its current location.

Task descriptions: We evaluate our proposed model on three different control tasks by modifying the MAPE [81].

1. **Rendezvous**: Each agent tries to reach its preassigned goal while avoiding collisions with other entities in the environment. The episode length of this task is 25, with a time step of 30 seconds, representing a simulation duration of 12.5 minutes

2. **Cluster Formation**: There is a single landmark, and the agents try to position themselves in an N-sided regular polygon with the landmark at its center. The episode length of this task is 30, with a time step of 30 seconds, representing a simulation duration of 15 minutes

3. **Trailing Formation**: There are two landmarks, and the agents try to position themselves equally spread out in a line between the two. The episode length of this task is 30, with a time step of 30 seconds, representing a simulation duration of 15 minutes

Figure 3.2 provides a visual of the three environments. We first focus on the *Rendezvous* task environment in sections 3.7.1 and 3.7.3, and 3.7.5. The performance on the other tasks is shown in Section 3.7.4. We note that the episode length is the same for both training and evaluation. Longer time periods can be modeled in our framework by either increasing the time step within an episode dt, or increasing the overall episode length.

3.7 Results

3.7.1 Transfer learning

Our first experiment serves to demonstrate the effectiveness of transfer learning from a ground environment to a space environment when compared to other similar MARL algorithms. We compare against the best performing of several popular MARL algorithms: RMAPPO, VDN [96], RMADDPG [104], QMIX [95], and MATD3 [93].

RMAPPO - We adapt [94] to include a recurrent neural network (RNN) to mine useful information hidden in the historical observations, as done in [90, 105]. After extensive testing, we found that the addition of the RNN leads to stronger overall performance in the ground and space environments.

VDN - We add a recurrent neural network to the original implementation of Value Decomposition Networks[96].

RMADDPG - We use the RNN adaptation of MADDPG as introduced in [104]. Similar to RMAPPO, the addition of the recurrent neural network enables the model to learn time dependencies across observations and communication.

When determining which of the three baselines would represent a representative comparison, we tested their performance when using both local information, as with SpaceMARL,

or global information. When an algorithm is using local information, each satellite's observation is only its own state and goal, $o_{loc}^{(i)} = [p^{(i)}, v^{(i)}, p_{goal}^{(i)}]$. By comparison, in a global information structure, a satellite also has access to the position of all other agents in the environment, $o_{glob}^{(i)} = [p^{(i)}, v^{(i)}, p_{goal}^{(i)}, p_{other}^{(i)}]$, where $p_{other}^{(i)}$ comprises the relative positions of all other entities in the environment.

Algorithm Name	**Information Mode**	**Required Communication Between Satellites**
Global RMAPPO	Global	Complete
Local RMAPPO	Local	None
VDN	Global	Complete
RMADDPG	Global	Complete
DGN	Local	Neighborhood
SpaceMARL	Local	Neighborhood

Table 3.1: The type of information that is passed between satellites, and the assumptions on the amount of communication that occurs for each baseline.

The specific information modes of each baseline algorithm, and the amount of message passing assumed between agents are shown in Table 3.1. The 'Required Communications Between Satellites' column indicates how many other satellites the satellite will communicate with at each time step. A designation of 'Complete' means that each satellite must communicate with every other satellite at each step. A designation of 'None' means that the satellite does not communicate with any other satellite during the whole episode. A designation of 'Neighborhood' means that the satellite is communicating with some subset of satellites within its sensing range.

Figure 3.3 shows the training performance of SpaceMARL and the aforementioned baselines. The transfer learning model was initialized using weights trained on a ground environment and then trained in the space environment. Consequently, in Figure 3.3, we offset the plot of SpaceMARL with transfer learning by 200,000 steps (the number of steps used to train the ground-based model). We trained each algorithm. The shaded area envelops one standard deviation of the runs.

Three of the baseline methods (RMAPPO [94], VDN [96] and RMADDPG [104]) achieved their best performance when using global information. We also explored the performance of QMIX [95] and MATD3[93], which are excluded from the figure as their performance was poor, and required significantly longer training times to reach rewards comparable to VDN and RMADDPG even when using global information. While the assumption of global information sharing can help determine a performance bound, it is not realistic in practice. By contrast, SpaceMARL (both with and without transfer learning) uses only local information, and requires satellites to only communicate when they are within sensing proximity of one another. Over the training period, SpaceMARL with transfer learning reaches a similar reward to the global RMAPPO [94], despite needing less information. This finding indicates that the quality of the information, rather than its quantity, is an important driver of performance. We also compare SpaceMARL's performance against a variant of RMAPPO that uses local information. In this local RMAPPO structure, each satellite only has access

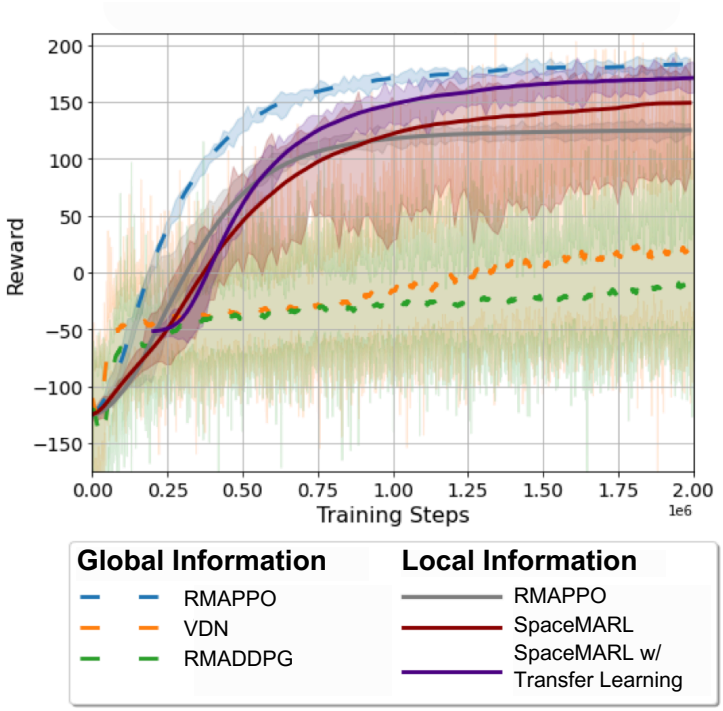


Figure 3.3: SpaceMARL performance against comparable algorithms in the Space Environment.

to its own observations. Therefore, the performance of local RMAPPO sets a minimum performance bound for the RMAPPO algorithm, illustrating a case where satellites are not cooperating with one another. Since SpaceMARL outperforms local RMAPPO, our approach can strike a balance between the communication requirement of global RMAPPO, and the independence of local RMAPPO. This further validates that our methodology is able to synthesize locally available information meaningfully and perform better than if the satellites had operated independently.

Furthermore, we see in Figure 3.3 that SpaceMARL with transfer learning outperforms the SpaceMARL model that was trained from scratch on the satellite environment. When learning from scratch, SpaceMARL needs to learn both the dynamical model and the reward optimization by learning how to get to the assigned goal. Through transfer learning, the model is able to focus on learning the dynamics, because the reward optimization process is already learned to some extent in the prior ground model. Another benefit of the transfer learning model is that it is not vulnerable to seed initializations. Through training, we observe that certain seeds have better initializations, which help the algorithm to learn faster and, therefore, better. By using the weights from a ground model, the transfer learning model is not dependent on these random seed weight initializations. Because of this, the standard deviation of the transfer learning is significantly smaller than classical SpaceMARL. These results indicate that ground-based models can be used to accelerate training for space traffic applications.

Based on the above results, the benefits of using SpaceMARL with transfer learning for such space applications are mainly two-fold: (1) The simplicity of the ground dynamics and

the availability of more established ground simulation environments make transfer learning from ground to space an attractive approach, and (2) SpaceMARL with transfer learning appears to be less susceptible to the performance of the random seeds used in training.

3.7.2 Scalability

Train \ Test		m= 3	m= 5	m= 10
n= 3	Reward/m	61.57	60.21	57.78
	T	0.44	0.44	0.43
	(# col)/m	0.36	0.77	1.41
	S%	98	94	96
n= 5	Reward/m	60.52	60.52	57.07
	T	0.44	0.44	0.44
	(# col)/m	0.78	1.28	1.41
	S%	98	98	91

Table 3.2: Performance metrics obtained by training SpaceMARL on a space environment with n satellites and testing it on one with m satellites: (a) Total reward obtained in an episode per agent, Reward/m. (b) Fraction of episode taken on average by agents to reach their goal, T (lower is better). (c) Average number of collisions per agent in an episode, #col/m (lower is better). (d) Success rate, S%: percentage of episodes in which all agents can get to their goals (higher is better).

The first experiment, shown in Table 3.2, demonstrates the scalability of our algorithm when trained on n agents and tested on m agents. The number of obstacles is held constant to be 3 throughout both all training and evaluation. When evaluating the connectivity throughout the experiments, we found that each agent maintained a connection with at least one entity in the environment more than 90% of the time, meaning that the vast majority of the experiment involves information-sharing between entities. We find that in all scenarios considered, our approach can control the satellites to reach their goals within approximately 44% of the episode length. As expected, the number of collisions per agent increases when there are more satellites (i.e., the environment has become more dense). As mentioned in Section 3.5, these values can be improved further by modifying the reward function or by using control barrier functions.

A key finding is that the reward per agent remains approximately the same even when the model is trained with $n < m$. Our approach also has a high success rate for unseen scenarios, as evidenced by its success rate.

3.7.3 Sensitivity to perturbations

Table 3.3 demonstrates the performance of SpaceMARL with transfer learning in the perturbed satellite environment for 3 agents and 3 obstacles on the *rendezvous* task. The transfer learning model was initialized using weights trained on a ground environment and then trained in the perturbed space environment model for each specific inclination. To produce the results in the table, we ran 100 evaluation scenarios.

We rely on several metrics to analyze the performance of our algorithm across 100 test episodes:

1. **Average Reward:** The total rewards obtained by the agents during an episode. A higher value corresponds to better performance.

2. **Success Rates:** Percentage of test episodes in which all agents are able to get to their goals, denoted $S\%$. A higher value corresponds to better performance.

3. **ΔV per agent:** The average ΔV in (m/s) each agent uses per episode, averaged across 100 episodes. We note that the action space allows for the satellites to take null actions. These null actions are not included in the Δ V, only movements in the $\pm$ x and $\pm$ y directions. A lower value corresponds to better fuel efficiency.

We find that even in perturbed environments, the algorithm learns effectively, as demonstrated by the high success rates and average reward values. The total amount of ΔV per agent remains roughly the same in each episode, indicating that the evaluation scenarios themselves had little variation across inclinations. We also tested the transfer learning model on perturbed satellite environments with different inclinations than what it was trained on and found that it performed similarly.

Inclination (ϕ)	**0°**	**28°**	**45°**	**54°**	**63°**	**72°**	**81°**	**90°**
Average reward	170.10	166.33	167.07	169.73	168.14	171.78	174.29	169.76
Reward Standard deviation	$\pm$ 6.32	$\pm$ 6.18	$\pm$6.09	$\pm$ 3.07	$\pm$ 8.85	$\pm$ 5.45	$\pm$ 3.02	$\pm$ 6.10
Success Rates	97	99	100	98	98	98	97	97
Δ V per agent	0.19	0.19	0.19	0.22	0.18	0.18	0.19	0.18

Table 3.3: Sensitivity of SpaceMARL with transfer learning to different perturbations.

3.7.4 Performance in Other Task Environments

We then tested the effectiveness of our algorithm in more tasks that require more intricate coordination. In the *Rendezvous* task environment, coordination amongst agents was required only for collision avoidance (both with other agents and obstacles) since the goal positions for all the agents were predetermined. For the *Cluster Formation* and *Trailing Formation* tasks, the agents not only need to coordinate for collision avoidance but also need to develop consensus on their goals. Table 3.4 shows the success rate, the fraction of episodes taken to complete the task, Δ V, the number of collisions, and distance traveled per agent in the *Cluster Formation* and *Trailing Formation* tasks across 100 episodes. Global and local RMAPPO are used as a point of comparison, as RMAPPO does not use the GNN structure, and can better illustrate the performance of our algorithm.

In addition to the metrics used in Table 3.3, we also rely on several additional metrics to characterize the performance of our algorithm across 100 test episodes:

1. **Collisions:** The total number of collisions (both agent-agent and agent-obstacle) that agents had in an episode. The lower the metric the better the performance of our algorithm.

2. **Distance traveled per Agent:** The overall distance an agent travels to get to its goal, in meters. A lower value corresponds to more efficient path planning.

We note that all results presented in Table 3.4 are from evaluation, meaning that the algorithms no longer receive critic feedback and are relying on fixed weights to determine performance. As shown by Table 3.4, SpaceMARL is still able to achieve a success rate of relatively high success rate across all scenarios in the different environments while taking a similar fraction of the episode to complete as global RMAPPO. For the *cluster* and *trailing* tasks, the performance of SpaceMARL could be further improved through additional reward tuning parameters that emphasize the importance of distance between agents.

Task	**Agents**	**Obstacles**	**Metric**	**Algorithm**		
				Global RMAPPO	**Local RMAPPO**	**Space MARL**
Rendezvous	3	3	T	0.40	0.83	0.44
			$S\%$	100	39	98
			ΔV per Agent	0.22	0.15	0.19
			Collisions	0.46	4.60	0.38
			Dist Traveled per Agent	1297.33	854.70	1275.66
Cluster Formation	4	1	T	0.39	0.78	0.50
			$S\%$	98	52	74
			ΔV per Agent	0.16	0.27	0.16
			Collisions	0.07	0.35	0.28
			Dist Traveled per Agent	1231.33	645.35	1426.66
Trailing Formation	3	2	T	0.41	0.75	0.54
			$S\%$	100	47	83
			ΔV per Agent	0.20	0.13	0.15
			Collisions	0.33	3.89	0.36
			Dist Traveled per Agent	1333.33	1346.66	1469.0

Table 3.4: Performance of Global RMAPPO, Local RMAPPO and SpaceMARL on the *rendezvous*, *cluster formation*, and *trailing formation* tasks.

The ΔV per agent and distance traveled per agent metrics reveal more about the differences in performance between Global RMAPPO and SpaceMARL in the *cluster* and *trailing* tasks. SpaceMARL travels a further distance over the course of an episode but uses a similar amount of ΔV, indicating that the timing of the algorithm's action selection led the agents to move further over the course of the episode. SpaceMARL did not compensate for this increased distance with additional actions that led the satellites to reach their goal, leading to lower success rates in both tasks.

Regardless of task, local RMAPPO performs worse than SpaceMARL. In the local RMAPPO structure, each satellite only has access to its own observations and can be considered a minimum performance bound where satellites learn to navigate without communication from other satellites.

While still not as effective as SpaceMARL, local RMAPPO achieves lower collision rates in the *cluster* task over the *rendzevous* and *trailing* tasks. This reduction in collision rate indicates that the path planning involved with this task involves few satellite-to-satellite

interactions. In the *rendzevous* and *trailing* tasks, local RMAPPO performs significantly worse, resulting in a significantly higher number of collisions than with global RMAPPO and SpaceMARL. These findings indicate that in these tasks, there is some value in satellite-to-satellite communications to reduce collision.

When comparing SpaceMARL's performance on the three tasks, it performs the best on the rendezvous task. The current reward function prioritizes goal-reaching, rather than more nuanced movements like approximate positioning.

3.7.5 Value of sharing goals

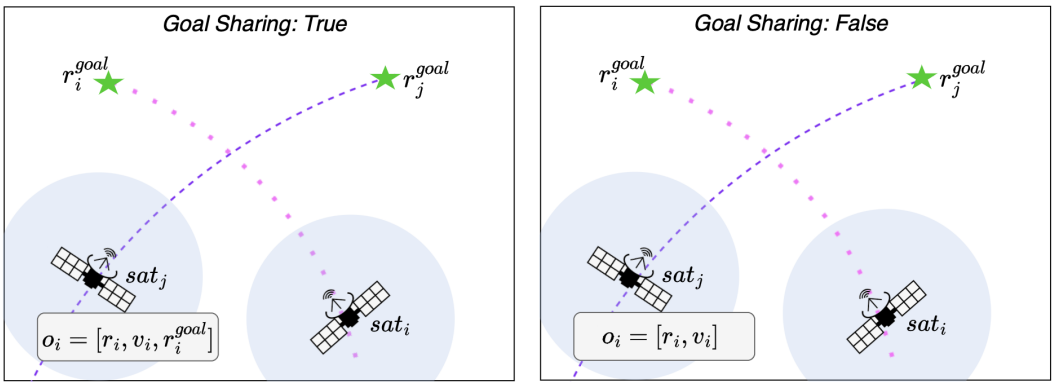


Figure 3.4: The agents in the goal sharing case share their intended final location

We use our model to evaluate the value of satellites sharing goal information (i.e., sharing their orbits, and any associated changes) with each other. To do this, we consider the SpaceMARL model that was trained from scratch in the space environment. We alter the previous experimental set-up to focus on $n = 2$ satellites. We initialize the satellites such that they both have the same distance to travel to their respective goals. Furthermore, their respective initial positions and goals are set so that their most direct path from the start to the goal intersects with the other satellites midway through their path, as shown in Figure 3.4. We selected the episode length and the minimum distance between a satellite's starting position and its goal to ensure that both satellites had sufficient time to make it to their respective goals, but not so much time that wait-and-see behaviors could emerge in the absence of information-sharing.

To focus on the information learned directly from sharing, we compare two cases: the first in which the satellites are sharing their goal information with each other (the *sharing* case), and the second in which they are not (the *hidden* case). Goal-sharing can be thought of as a proxy for the future movements of the satellite, as it will seek to move towards the goal over the course of an episode.

We evaluate four metrics to demonstrate the difference in performance as a result of sharing:

1. *Collision Difference*: For each scenario, denoted i, we compare the difference in the number of collisions in the hidden case h_i to that of the shared case s_i. A positive value indicates that there are more collisions in the hidden case.

2. *Distance Remaining*: We devised this metric to assess how close the satellites were to the goal at the end of the episode, in each of 100 random scenarios. We use the following equation to determine the distance remaining ratio.

$$\text{Distance Remaining} = \frac{h_{i,\text{final}} - s_{i,\text{final}}}{\text{Shortest Path}} \tag{3.11}$$

$h_{i,\text{final}}$ and $s_{i,\text{final}}$ represent the sum of the distances of the two satellites to their respective goals at the end of the episode for the hidden and sharing cases in scenario i, respectively. We normalize this value using the shortest possible paths that the satellites could have taken to the goal. A positive value indicates that at the end of the episode, the satellites are farther away from the goal in the hidden case than in the goal-sharing case.

3. *Distance Traveled*: This metric is used to assess the distance traveled by the satellites in the hidden case compared to the sharing case. $s_{i,\text{sum}}$ and $h_{i,\text{sum}}$ represent the sum of the total distances traveled by each satellite during scenario i. We rely on the following equation to scale the difference of distance traveled difference by the shared case. Here, a positive value indicates that the hidden cases had to travel further than the shared cases.

$$\text{Distance Traveled} = \frac{h_{i,\text{sum}} - s_{i,\text{sum}}}{s_{i,\text{sum}}} \tag{3.12}$$

4. ΔV *Difference*: This metric was used to assess the difference in fuel usage between the hidden and sharing cases. In our reinforcement learning model, each satellite has the option to exert a control action from a discrete set $[0, +x, -x, +y, -y]$. The magnitude of this control action is fixed. To calculate the differences in ΔV value, we compared the number of non-zero control actions taken by the hidden and shared satellites for each scenario across 100 scenarios. A positive value indicates that the hidden case had more control actions, and thus more ΔV than the shared case.

	Metric			
	Number of collisions	**Distance remaining to go**	**Distance traveled**	ΔV
Average Improvement	13.0%	26.8%	79.0%	-16.5%
Standard Deviation	81.0%	182.0%	132.0%	161.8%

Table 3.5: Percentage improvements through goal information-sharing, results are averaged over 100 evaluation episodes.

Table 3.5 shows our results. Through goal-sharing, we were able to achieve 13% lower collision rates on average, along with 26.8% and 79% improvements in the remaining distance to goal and distance traveled, respectively. These benefits were achieved at the expense of only a 16.5% increase in the ΔV. While the overall number of collisions in our model can be addressed through the inclusion of additional collision avoidance features like learned control barrier functions or artificial potential functions, the improvements in path planning through sharing are particularly significant. While performing these experiments, we also evaluated

the reward rates but found that sharing had negligible effect. We hypothesize that this is because the hidden case is able to reach the goal and thus receive the goal-reaching reward in the majority of cases. In our reinforcement learning framework, the goal-reaching reward is significantly higher than the penalties that occurred while currently not at the goal. As a result, when the hidden satellites successfully reach their goal, they are able to achieve a reward evaluation similar to the one they would have in the sharing case.

We note that this result possesses high variability, as demonstrated by the standard deviation across all four metrics. Upon further examination of the 100 scenarios, we attribute this variance to the environment's sensitivity to action selection, and the algorithm's threshold for goal reaching. When examining the cases that had higher ΔV usage, we found that SpaceMARL tends to over-compensate with additional non-zero actions when it has missed the goal. This results in increased ΔV usage, and exacerbates the distance between the satellite and its goal. As a small set of cases have this emergent behavior, this further stratifies the standard deviation. Another explanation for the high variance of these results is the threshold for goal reaching itself. There exists a subset of solutions that nearly reach the goal, but are not close enough. In this subset, SpaceMARL tends to overcorrect to get near the goal again, resulting in a worse solution than the one that nearly reached the goal. This behavioral subset results in higher ΔV usage and distance remaining to the goal, thus creating a wider spread of possible results. The behavioral pattern of over-correcting via more non-zero actions can be attributed to our action space being discretized rather than continuous. Due to this, the algorithm is unable to select the magnitude of acceleration it should apply, and instead must repetitively use set u_x and u_y parameters. Future work will explore using a continuous action space, and more sophisticated control methodologies.

3.7.6 Heterogeneous Agents

Next, we use our model to evaluate the impact of the individual actors themselves (i.e., who is sharing). We perform an experiment to assess the impact of actors with different levels of knowledge on the resultant satellite coordination. As mentioned in section 3.3, we rely on a satellite entity graph encoding, where entities are separated into three categories $\texttt{entity\_type(j)} \in \{\texttt{satellite}, \texttt{obstacle}, \texttt{goal}\}$. We alter the experimental set-up to focus on $N = 3$ satellites performing a *rendezvous* task, where each of the satellites must reach a goal location. We initialize the $N = 3$ satellites such that 1 of the 3 satellites has reduced observation abilities, and thus the observation of its state contains a Gaussian noise parameter $o_i = o_i + \texttt{R}$, as shown in Figure 3.5.

Table 3.6 shows the results of the sensitivity study of different σ_{max} values, across 100 evaluation scenarios. For each noise value, σ, we trained a model in the space environment with the corresponding noise level added to the agent's observation abilities. Then, we evaluate the performance of these trained models across several metrics. First, we calculate the average reward across all 100 scenarios, as well as the standard deviation of the reward parameters. Next, we calculate the sucesss rate, which is the number of times all three agents successfully went to their goal out of a 100 cases. Finally, we evaluate the number of control actions used throughout an episode on average for each agent. This metric is used to determine how much satellite's need to maneuver to get to get to their goal.

We note that across all σ parameters, SpaceMARL's performance possesses high variabil-

Noise (σ)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average Reward	33.50	43.34	31.40	31.54	20.31	37.82	21.18	24.52	32.43	25.965
Reward Std	±75.10	±65.14	±78.16	±63.63	±67.67	±77.78	±65.40	±76.72	±71.12	±80.29
Success Rates	62	80	41	48	35	67	27	43	53	34
Δ V per agent	19	20	20	19	19	20	21	21	22	22

Table 3.6: Comparison of SpaceMARL's performance with different noise parameters across 100 evaluation scenarios.

ity across evaluation scenarios, as demonstrated by the reward standard deviation. When looking at trends for individual σ parameters, it is hard to determine if there is a clear relationship between the noise σ and the resultant success rates and reward amounts. The experiments with σ parameters in range [0.1, 0.5], achieve higher success rates on average (avg 53.2) than those than those in the larger sigma range [0.6, 1.0] (avg success 44.8). Similarly, the reward for the smaller σ values is also larger (avg 32.01 vs avg 28.38). Moreover, the general trend for the number of actions taken shows that as σ increase, so does the number of control actions taken over the course of the episode.

With the impact of σ explored, we evaluate the algorithms ability to interpolate with different classes of satellite (i.e. one base satellite class, and one injected with the noise parameter σ). As mentioned in section 3.3, we rely on a satellite entity graph encoding, where entities are separated into three categories $\texttt{entity\_type(j)} \in \{\texttt{satellite}, \texttt{obstacle}, \texttt{goal}\}$. We alter the experimental set-up to focus on $N = 3$ satellites performing a *rendezvous* task, where each of the satellites must reach a goal location. We initialize the $N = 3$ satellites such that 1 of the 3 satellites has reduced observation abilities, and thus the observation of its state contains a Gaussian noise parameter $o_i = o_i + \texttt{R}$, as shown in Figure 3.5.

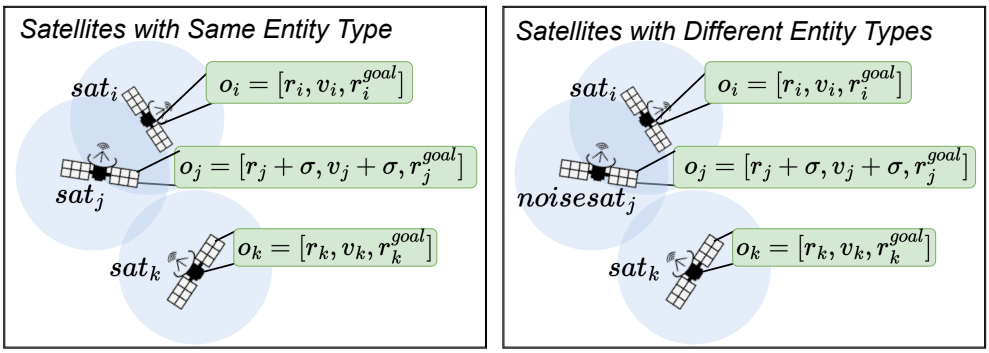


Figure 3.5: Two entity-graph formulations when additional class `noise_sat` is added

We model two graph entity embedding formulations. In the first formulation, every object in the environment is classified as $\texttt{entity\_type(j)} \in \{\texttt{satellite}, \texttt{obstacle}, \texttt{goal}\}$. As shown in the left pane of Figure 3.5, in this formulation, the noisy satellite has entity type `satellite`. These classifications are transformed into a one-hot encoding, where, for example, a satellite class would be represented as [1,0,0]. This one-hot encoding is used as an input in the graph transformer in the x_i^j variable (see Fig. 1). In this formulation, we wanted to test if SpaceMARL would learn to weigh individual nodes differently to compensate for two classes of satellites.

We compared SpaceMARL's performance in this first formulation against a second formulation where we now add another entity type encoding, *noiseSat* (as shown on the right-hand side of Figure 3.5). In the second formulation, we increase the number of categorizations so that every object in the environment is classified as `entity_type(j)` $\in$ {`noise_satellite`, `satellite`, `obstacle`, `goal`}. As shown in the right pane of Figure 3.5, in this formulation, the noisy satellite has entity type `noise_satellite`. These classifications are transformed into a one-hot encoding, where, for example, a noisy satellite class would be represented as [1,0,0,0]. In this formulation, we wanted to see if the existence of an additional classification type would help SpaceMARL learn to coordinate heterogeneous agents rather than just the graph weighting of the first formulation.

We rely on several additional metrics to assess the impact of heterogeneous agents with different quality of observations. First, we evaluate the average proportion of time over an episode that each satellite class is connected with each other (Average Connectivity of Base-Noise Satellites and Average Connectivity of Base-Base Satellites). Next, we evaluate how much each satellite class traveled over the course of the episode, in meters (Average Base Satellite Distance Traveled and Average Noise Satellite Distance Traveled). Then, we analyze the success rate of each class of satellite in getting to their individual goal (Base Satellite Success Rate, and Noise Satellite Success Rate). Finally, we evaluate the reward achieved in each case to assess SpaceMARL's overall strength in coordinating heterogeneous agents.

	Graph Type Class					
	Different Entity Types $\sigma = 1$	Different Entity Types $\sigma = 0.5$	Different Entity Types $\sigma = 0.1$	Same Entity Types $\sigma = 1$	Same Entity Types $\sigma = 0.5$	Same Entity Types $\sigma = 0.1$
Average Base Satellite Distance Traveled	1.49	1.29	1.22	1.23	1.27	1.30
Average Noise Satellite Distance Traveled	2.32	1.98	2.09	2.28	2.28	2.11
Base Satellite Success Rate	98.71	98.86	100	98.71	97.61	99.5
Noise Satellite Success Rate	14.89	11.36	14.63	17.94	10.1	15.55
Average Reward per Episode	113.28	112.59	107.01	105.84	108	97.2

Table 3.7: Difference in performance for alternate satellite-entity graph formulation.

Consistent with our previous experiments, we find that the average reward per episode is greater when the noise level σ is lower (as was seen in the *rendezvous* task). When evaluating the difference in performance between the two entity type formulations, we observe very modest improvements in the rewards, suggesting that additional `entity_type` definitions in the graph formulation can provide a marginal improvement in learning.

When evaluating the performance of the satellites on a class-specific level, the base satellites had a near-perfect success rate at the *rendezvous* task, whereas the `noise` satellites struggled to reach their goal with a success rate of roughly 15%. Furthermore, the overall path-planning abilities of the `noise` satellite class were limited due to their noisy obser-

vations; the average `noise` satellite class traveled nearly twice as far as the average base satellite class.

3.8 Chapter summary and contributions

This chapter discussed the development of a combined graph neural network multi-agent reinforcement learning algorithm for the control of a set of satellites in a close environment.

The initial part of this chapter focused on presenting our multi-agent reinforcement learning framework, SpaceMARL, highlighting the development of the space environment and the transfer learning model employed in this work. Then, we demonstrated the effectiveness of transfer learning by comparing SpaceMARL's performance in the space environment against several other popular MARL frameworks.

The next experiment evaluated the scalability of SpaceMARL, and we demonstrated that the algorithm is able to train on n agents and successfully evaluate on $> n$ agents. We then considered a more sophisticated variant of the environment that includes J2 perturbations and tested SpaceMARL's performance. Section 3.7.4 presented the performance of SpaceMARL across three tasks: *rendezvous*, *trailing formation*, and *cluster formation*. We compared SpaceMARL's performance against a global sharing model of RMAPPO to characterize the best-case performance where the satellites have perfect communication and sensing abilities.

With SpaceMARL's performance in the space environment established, we then performed several studies that used its graph structure to characterize the impact of different pieces of information and the impact of different actor characteristics.

The specific contributions of this chapter are the following.

Contribution 3.1	Development of a multi-agent reinforcement learning framework for space applications, named SpaceMARL
Contribution 3.2	Evaluation of the effectiveness of transfer learning from ground to space-based applications
Contribution 3.3	Ablation studies of the impact of different graph formulations on learned satellite coordination in space-based environment

Chapter 4

Asynchronous and Communication-Limited Multi-Agent Reinforcement Learning

4.1 Introduction

We consider the problem where a set of satellites are making a set of decisions asynchronously, and communication between them is limited. In extreme environments such as those underwater or in space, the frequency of communication between agents is often constrained [106, 107]. 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. We focus on an alternate framework to SpaceMARL that accounts for time differences between the actions and observations of different satellites.

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. 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.

In this chapter, we discuss the development of AsyncCoMARL, a graph transformer-based communication protocol for MARL that relies on dynamic graphs to capture asynchronous and infrequent communications between agents.

AsyncCoMARL 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 AsyncCoMARL use less communication and outperform other MARL methods. The main contributions of this chapter are:

1. We propose AsyncCoMARL, 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 AsyncCoMARL on two MARL benchmarks (Cooperative Navigation [76] and Rover-Tower[108]) and show that our method can achieve superior performance while using less communication.

In this chapter, we will first introduce our methodology in Section 4.2. We present our results in 4.3.2, and discuss key highlights and summarize the key contributions of this work in Section 4.6.

4.2 Methodology

4.2.1 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 4.1, $\tau_1^{(i)}$ represents the time of the first action that agent i has taken. In Figure 4.1 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.

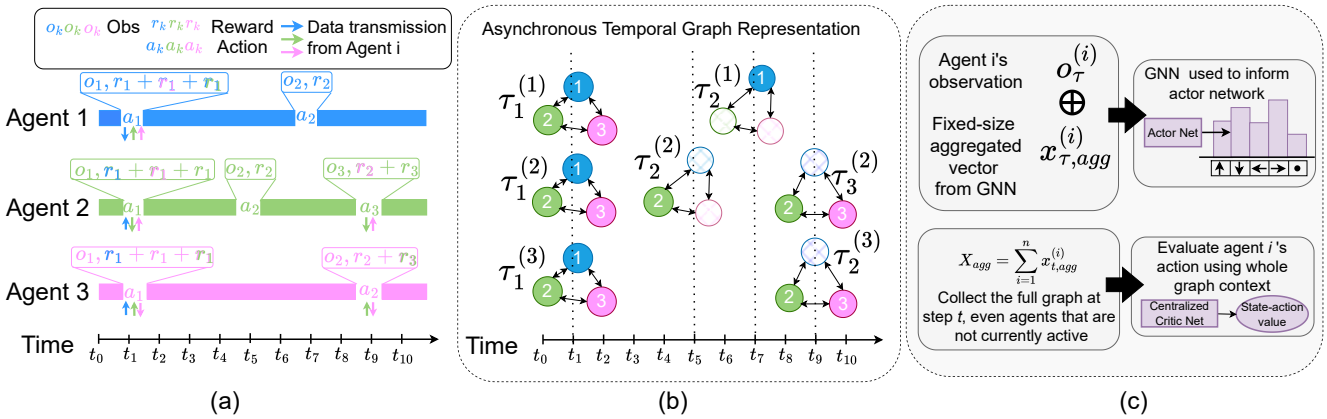


Figure 4.1: Overview of AsyncCoMARL: (a) Environment. Agents within our environment take actions and observations asynchronously. To encourage collaboration, when agents take actions at the same time t, they receive a shared reward. The sequence of actions and observations for agent i is referred by timescale $\tau^{(i)}$. The arrows indicate data transmissions, which represent the most recent graph observation $x^{(i)}_{\tau_a gg}$. 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 GNN, $x^{(i)}_{\tau,agg}$ and fed into the actor network. The critic takes the full graph representation X_{agg} and evaluates agent i's action.

4.2.2 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 [54], 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 $x_j = [p_i^j, v_i^j, p_i^{\text{goal},j}, \texttt{entity\_type(j)}]$ 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 4.1. 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.

4.2.3 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) [86] 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

$$x_i' = W_1 \cdot x_i + \sum_{j \in \mathbb{N}(i)} \alpha_{i,j} W_2 \cdot x_j \tag{4.1}$$

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

4.2.4 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

$$\mathcal{R}_\tau^{(i)}(s_\tau^{(i)}, a_\tau^{(i)}) = \mathcal{R}_{\text{dist}}(s_\tau^{(i)}, a_\tau^{(i)}) + \rho \mathcal{R}_{\text{goal}}(s_\tau^{(i)}, a_\tau^{(i)}) - \kappa C \tag{4.2}$$

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.

$$\mathcal{R}_{\text{total}}(s_\tau, A_\tau) = \sum_{i}^{\eta_{act}} \mathcal{R}_\tau^{(i)}(s_\tau^{(i)}, a_\tau^{(i)}) \tag{4.3}$$

4.2.5 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: $\langle \mathcal{N}, \mathcal{S}, \mathcal{O}, \mathcal{A}, R, P, \gamma, \mathcal{G} \rangle$. $g^{(i)} = \mathcal{G}(s; i)$ 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 4.1, 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 state-value function for policy π_{θ_i}.

$$V_\phi^{\pi_{\theta_i}}(s_\tau^{(i)}, a_\tau^{(i)}, X_{agg}) = \mathbb{E}_{a \sim \pi_{\theta_i}} \left[\sum_{\tau=0}^{T} \gamma^\tau \mathcal{R}_\tau | s_0 = s \right] \tag{4.4}$$

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:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\pi_{\theta_i}} [\nabla_{\theta_i} \log \pi_{\theta_i}(a_\tau^{(i)} | o_\tau^{(i)}, g_\tau^{(i)}) \cdot V_\phi^{\pi_{\theta_i}}(s_\tau^{(i)}, a_\tau^{(i)}, X_{agg})] \tag{4.5}$$

Algorithm 1 shows the complete training setup for AsyncCoMARL. As mentioned in Section 4.2.1, 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.

4.3 Experimental Results

4.3.1 Experimental Setting

We conduct experiments in two environments, Cooperative Navigation and Rover-Tower, specifically chosen to emulate real-world scenarios with communication constraints. These

Algorithm 1 AsyncCoMARL Training setup

Initialize a decentralized policy network for each agent i: $\vec{\pi}_{\theta_i}$
Initialize a centralized critic network $V_{\phi}^{\vec{\pi}_\theta}$
for $episode = 1 : M$ **do**
  Reset the environment
  Reset buffer $\mathcal{D} = \{\}$
  for step $t = 0 : T_f$ **do**
    if $\exists i \in \{1, .., n\}$ agent$^{(i)}$.status = active **then**
      $a^{(i)} \leftarrow$get action
      Update environment state with $a^{(i)}$
      $\mathcal{R}_{total}^{(i)} \leftarrow \sum_{j=1}^{\eta_{active}} \mathcal{R}_t^{(j)}$
      Collect $\langle o^{(i)}, a^{(i)}, g^{(i)}, r^{(i)} \rangle$ into buffer $\mathcal{D}$ at $\tau^{(i)}$
    else
      for $i \in \{1, .., n\}$ **do**
        $a^{(i)} = \emptyset$
        Update environment state with $a^{(i)}$
        $$\text{mask}(i) = \begin{cases} 1, & \text{if agent } i \text{ is finished} \\ 0, & \text{otherwise} \end{cases}$$
      end for
    end if
  end for
  Follow standard MARL update process
end for

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 tower. 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 [109]. 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: [76] propose a modified MPE environment that uses the satellite dynamics following the Chlohessy-Wiltshire model [99], 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 Chlohessy-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.

Rover-Tower: [108] 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.

4.3.2 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 [110], we use the success rate metric to indicate the overall success of the total number of agents.

4.3.3 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.

4.4 Motivating Experiment

First, we compare the performance of classical SpaceMARL in the asynchronous setting against our newer algorithmic variant AsyncCoMARL. We see in Table 4.1 that SpaceMARL fails to learn a policy due to the sparse inputs from in the asynchronous steps in our asynchronous environment. The inconsistency in each agent's action taking causes a significant performance difference, and causes the policies learned to be biased to not take any action at all. This results in lower overall collision rates due to the lack of agent movement. As AsyncCoMARL was designed with asynchronous time communications in mind, AsyncCoMARL is able to compensate for the sparse action space and attain a high success rate by comparison. This experiment motivates us to find a way in which sparse information can be leveraged to accommodate stringent communication budgets.

Metrics	AsyncCoMARL	SpaceMARL
Communication Frequency (f_{comm})	0.24	0.78
Success Rate ($S\%$)	92	0
Episode Completion (T)	0.22	1.0
Average Collisions (#col)	0.76	0.0

Table 4.1: Comparison of SpaceMARL against the asynchronous variant, AsyncCoMARL.

4.5 Comparison of AsyncCoMARL with Other Methods

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 [111]	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 [112]	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 [108]	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 [110]	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 [113]	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 [52]	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
AsyncCoMARL	**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 4.2: Comparison of AsynCoMARL with other baseline methods for scenarios with 3, 5, 7, and 10 agents in the Cooperative Navigation environment.

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** [111]: 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** [108]: 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** [112]: The authors extend multi-agent proximal policy optimization [94] to the asynchronous setting and apply an invariant CNN-based policy to address intra-agent communication.
- **TransfQMix** [110]: 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** [113]: 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** [52]: DGN relies on graph convolutional networks to model the relational representation and implicitly models the action coordination.

Performance on Cooperative Navigation Environment

Table 4.2 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.

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.

Performance on Rover-Tower Environment

Table 4.3 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

Algorithm	Metrics		
	$f_{\text{comm}} \downarrow$	$T \downarrow$	$S\% \uparrow$
Actor-Attention-Critic [108]	0.21	0.84	56%
AsyncMAPPO [112]	0.24	0.98	0%
TransfQmix [110]	0.40	0.98	0%
AsyncCoMARL (our method)	**0.14**	**0.55**	**50%**

Table 4.3: Comparison of AsynCoMARL with other baseline methods for scenarios with 4 rovers and 4 towers in the Rover-Tower environment.

two classes, Actor-Attention-Critic creates a separate network for the rover class and the tower class, whereas AsyncCoMarl 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.

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 4.2, 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 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.

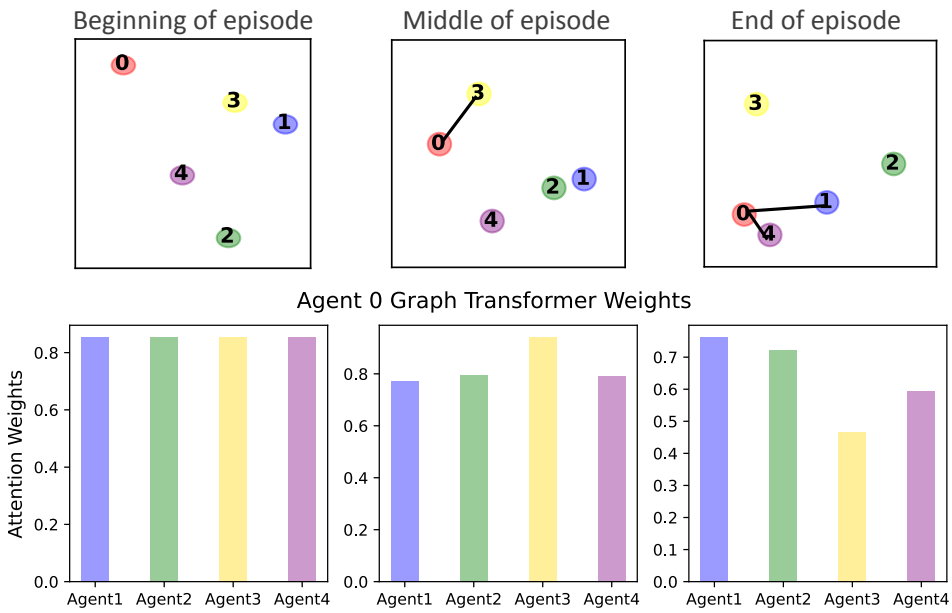


Figure 4.2: Attention weights for agent 0 in the $n = 5$ agent Cooperative Navigation task. We compare the changes in graph transformer attention at three discrete periods during the episode at the beginning, middle, and end.

4.5.1 Ablation Studies

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) AsyncCoMARL, 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 4.4. We aimed to determine whether there was a relationship between an agent's

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

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.

Reward Formulation

In developing AsyncCoMARL, 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 SpaceMARL algorithm [74]. For a single agent, the reward is calculated by Equation 4.2. In this formulation, the goal-reaching reward, $\mathcal{R}^{(i)}_{goal,t} = 5$, 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, $\mathcal{R}^{(i)}_{total}$. 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. Using repeated rewards at the goal is intended to provide a stable signal, reinforcing the learning process.

- **Piecewise Reward**: In the piecewise structure, each agent i receives a $\mathcal{R}^{(i)}_{goal,t}$ value of $+5$ at the first time step that it reaches gets within distance δ of the goal, p^{goal}_i. For every time step after, the goal-reaching value is changed in Equation 4.2 to be a smaller value.

$$\mathcal{R}^{(i)}_{goal,t} = \begin{cases} +5 & t_g = ||s^{(i)}_t - p^{goal}_i|| \leq \delta \\ +0.5 & t > t_g \end{cases} \tag{4.6}$$

  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 $\mathcal{R}^{(i)}_{goal,t}$ 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.

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

- **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).

Table 4.5 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.

4.6 Chapter summary and contributions

This chapter discussed the development of a graph-transformer communication protocol for asynchronous multi-agent reinforcement learning, which is 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 AsyncCoMARL'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. The specific contributions of this chapter are the following.

Contribution 4.1	Development of an asynchronous multi-agent learning framework, AsyncCoMARL, designed to address minimal communication for a set of independent, locally communicating agents
Contribution 4.2	Empirical comparisons of different attention mechanisms and their relevancy for the asynchronous coordination problem in multi-agent reinforcement learning

Chapter 5

Game Theoretic Models of Non-Cooperative Satellite Interaction

5.1 Introduction

Cooperative and competitive proximity operations for satellites are deeply interconnected, as both involve strategic planning and maneuvering within shared orbital spaces. Cooperative operations, such as docking or formation flying, emphasize collaboration and require precise coordination, which can inform strategies for avoiding conflict in competitive scenarios. Conversely, insights from competitive operations, like collision avoidance in adversarial environments, can enhance autonomous decision-making and risk assessment in cooperative missions, ensuring safety and efficiency. This chapter examines non-cooperative interactions between satellites. We apply game theory, which uses mathematical models to analyze strategic interactions between rational decision-makers and predict outcomes in competitive and cooperative scenarios. In the context of satellite collision avoidance, we apply game-theoretic approaches to evaluate decisions about when and whether to maneuver to avoid potential collisions, factoring in the differing risk thresholds of satellites.

Our contributions include the following:

- A new one-shot game framework to model decision-making for non-cooperative satellite collision avoidance
- The analysis of the relationship between different satellite characteristics and conjunction parameters to study the emergence of strategic behaviors in satellite collision avoidance.
- The design of a repeated game framework that integrates orbital estimation techniques to propagate the covariance over the decision period.

In this chapter, we will discuss the development of a game theoretic model to study competitive satellite interactions. This chapter is based on work that was presented in [114]. First, we provide a brief introduction to game theory and our one-shot game in Sections 5.2 and 5.3, and provide a motivating experiment to build intuition in Section 5.4. Then, we discuss the mechanics behind our model in Section 5.5, introducing the state

estimation functions, satellite risk thresholds, and optimization method. We validate our state estimation and covariance propagation in Section 5.6. In Section 5.7, we show the results of our approach and summarize the takeaways of this chapter in Section 5.8.

5.2 Game Theory Overview

Game theory studies the interactions among rational, strategic agents. These agents are capable of reasoning about their own and others' actions to make decisions that maximize their utility or reward [115].

Many games may be written in matrix, or normal form, where n players each have a set of actions to choose from. In a two-player matrix game, the rows represent the actions of the first player, the columns represent the actions of the second player, and the entries indicate the utility for each player based on the combination of actions, known as the payoff matrix [116].

The game of chicken is a well-known model of conflict between two players. Imagine two drivers heading toward each other in a head-on collision. If both continue straight, they will crash, and the only way to avoid a collision is for at least one to swerve. However, if one driver swerves while the other goes straight, the one who swerves is considered chicken and loses, while the driver who goes straight wins. We assume identical drive reaction times and abilities of cars. The decisions are taken simultaneously and cannot be revoked [117].

Table 5.1 gives the payoff matrix for the game of chicken.

		Agent 2	
		Swerve	Straight
Agent 1	Swerve	$(0, 0)$	$(-1, 1)$
	Straight	$(1, -1)$	$(-10, -10)$

Table 5.1: Payoff matrix for the game of chicken.

Consider the best responses for agent 1 in the game of chicken, which are the actions that minimize agent 1's cost for each action of agent 2. If agent 2 goes straight, agent 1 should swerve, since swerving results in a cost of 1, which is less than the cost of 10 for not swerving. However, if agent 2 swerves, agent 1 should continue straight, as going straight results in a cost of -1, which is less than the 0 cost of swerving. This produces an established phenomenon in game theory known as Nash equilibrium, where each player is making the best possible move given the circumstances. In other words, each player's strategy is the best response to the strategies of the other players, and no one has an incentive to deviate from their chosen strategy [115].

As swerving is the best response to the other agent going straight, and vice versa, there are two Nash equilibria: one in which agent 1 swerves and agent 2 goes straight, and one in which agent 1 goes straight and agent 2 swerves. These are both pure strategy Nash equilibria. A pure strategy Nash equilibrium is a situation in a game where each player selects a single strategy (eg. swerve or straight), and no player can improve their payoff by unilaterally changing their chosen strategy.

In addition to the two pure strategy Nash equilibrium in the game of chicken, there exists a third class of Nash equilibria known as mixed strategy Nash equilibria Mixed strategy Nash equilibria exist because, in some games, players can achieve optimal strategies by randomizing their choices rather than selecting one strategy 100% of the time. In the mixed strategy case, each player selects their strategy according to a specific probability distribution that makes the other players' strategies optimal [118]. Many real-world scenarios involve uncertainty and incomplete information, where players cannot predict their opponents' choices with certainty. Mixed strategies allow for probabilistic decision-making, making the model more realistic and applicable to diverse situations. Furthermore, in many practical situations, observed behaviors don't align with pure strategies. Mixed strategies can help explain behaviors like variability in pricing strategies, player movement in sports, or military tactics.

Let us assume that each agent chooses to swerve with probability p. By symmetry, this strategy is identical for both players. To find the probability, p, that yields a Nash equilibrium, we can analyze the best responses. Specifically, we can find the p that causes each agent to be indifferent to swerving or going straight, meaning both actions yield the same costs.

The expected cost of swerving in response to the other agent swerving with probability p is $0p+1(1-p) = 1-p$, while the expected cost of going straight is $-1p+10(1-p) = 10-11p$. Setting these equal to find the p that leads to indifference and thus a Nash equilibrium yields $1 - p = 10 - 11p$, so $p = 9/10 = 0.9$. Thus, a strategy for both players of swerving with probability 0.9 and going straight with probability 0.1 is a mixed strategy Nash equilibrium.

The Nash equilibrium, whether pure or mixed, explains how individuals or entities make decisions by considering others' actions, leading to stable outcomes where no one benefits from changing their strategy alone. These stable states are valuable for ensuring predictability, improving decision-making, and optimizing resource use in areas like economics, politics, and engineering.

5.3 Game Framework

We adapt the game of chicken to the satellite collision avoidance problem. In this variant of the game, we aim to account for the fact that the decision to maneuver depends on the satellite's risk threshold and their expected estimate of the future cost of moving.

We propose a 2 player game involving two satellites, also referred to as agents. We formulate the game so that each player has the following traits, where $i = \{1, 2\}$:

- Action $a_i^t \in \{wait, maneuver\}$ at timestep t, with $a^t = \{a_1^t, a_2^t\}$. $a_i = 1$ indicates a maneuver, and $a_i = 0$ is for waiting. When satellites maneuver, they apply a thrust in the direction that increases the distance between satellites.

- Type θ_i^t represents the satellite's risk threshold. This type of variable is independent of, and known to, the other satellite.

- Probability of collision from satellite's perspective: $p(|\hat{s}[T]| \leq \gamma)$. This represents the satellite's current estimate of the probability of collision.

- Payoff $u_i^t : A^t \times S^t \times \phi_i \mapsto R$. This represents the payoff at each round of the game, dependent on the action taken, the state outcome, and the type of agent.
- Strategy $\pi_i : \theta_i^t \times S^t \mapsto A_i^t$. A recommended action is based on the risk threshold of the satellite and the relative distance between the two satellites.

Then, we use the following notation to describe the conjunction between the two satellites.

- Time of closest approach (TCA), occurs at final timestep T.
- Distance at closest approach (DCA), denoted $s[T] \in R$.
- State space S. In our problem, the state represents the relative distance between the two satellites at the time of closest approach, so $S \in R^3$. Final state is $s_T \in S$.
- $p(\hat{s}[T])$ Probability of collision for the time T. We apply the Foster and Estes formulation to calculate the probability of collision [119]. A more detailed discussion of this calculation is included in Section 5.5.3.

Using this information, we can define the payoff to each satellite in the payoff matrix. The key idea behind our game framework is that a satellite's payoff depends on the probability of collision. We formulate the game such that if the probability of collision is above the satellite risk threshold, then the satellite should receive a highly negative outcome. If it is beneath the satellite risk threshold, then the satellite is neutral. We model this in the terminal cost function h_i, where H is a very large number:

$$f_i(p(|\hat{s}[T]| \leq \gamma) = \begin{cases} 0 & \text{if } p(|\hat{s}[T]| \leq \gamma) \leq \theta_i[t] \\ -F & \text{o.w} \end{cases} \tag{5.1}$$

We then model the cost of maneuvers for each satellite $g_i(a_i[t], t) \forall t = \{t_0, \ldots, T-1\}$. If the satellite doesn't maneuver, $a_i[t] = 0$, and the cost to a satellite is 0. Otherwise, if $a_i[t] = 1$, the satellite incurs a slight negative cost proportional to the maneuver used to get a constant change in $\hat{s}[t]$. This negative cost increases with time to reflect the fact that reduced planning time for satellite maneuvers impedes the satellite's ability to plan more fuel-optimal maneuvers, resulting in more rudimentary solutions. In our model, as TCA gets closer, more thrust is required to get a corresponding change in $\hat{s}[t]$. We model this increase in cost by an exponential function:

$$g_i(a_i[t], t) = -G_1 e^{G_2 t} a_i[t] \tag{5.2}$$

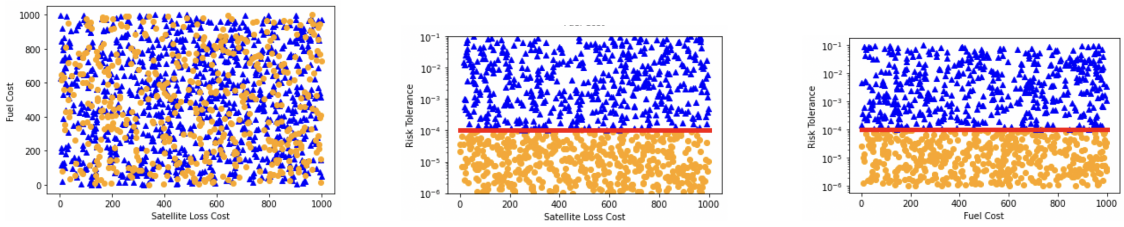
		Satellite 2	
		Move	Wait
Satellite 1	Move	(g,g)	(g,0)
	Wait	(0,g)	(f,f)

Table 5.2: Payoff matrix for the satellite collision avoidance problem.

The complete payoff matrix is shown in Table 5.2.

5.4 Motivating Experiment

We use this payoff matrix to examine how satellite risk threshold, maneuver costs, and terminal costs influence the Nash equilibrium.



(a) Fuel Cost vs Terminal Cost

(b) Satellite Risk Threshold vs Terminal Cost

(c) Satellite Risk Threshold vs Fuel Cost

Figure 5.1: The impact of the fuel cost, G, terminal cost H, and satellite risk threshold on resultant action selection. A blue triangle indicates waiting, while an orange circle indicates moving. The red line indicates the probability of collision for the wait-wait scenario. This factor has the largest impact on a satellite's action selection.

In Figure 5.1, we transform the 3D space containing the fuel cost, terminal cost, and satellite risk threshold onto three 2D planes. Figure 5.1a shows the fuel and terminal cost, and there is no clear correlation between the two parameters. Figures 5.1b and 5.1c present an alternate perspective. The separation of strategies by the red line highlights that the probability of collision when neither satellite swerves is the most significant factor influencing satellite strategy. If the probability of collision is beneath the satellite risk threshold, then the satellite does not maneuver (shown by the blue triangle). Conversely, if the probability of collision is greater than the satellite risk threshold, then the satellite will maneuver (shown by the orange circle). Thus, the most significant parameter in satellite decision-making is the probability of collision when both satellites don't maneuver.

Figure 5.2 and Table 5.3 examines how the relative ratio of long-term to short-term costs (H/G) affects the emergence of a mixed-strategy Nash equilibrium, where at least one player adopts a randomized strategy. For this evaluation, both satellites had a risk threshold that was less than the probability of collision if both satellites wait, $\theta < p_{wait\_wait}$.

Figure 5.2 demonstrates that as the ratio of long-term to short-term cost increases, the mixed Nash equilibrium strategy distribution favors moving. Although the one-shot game

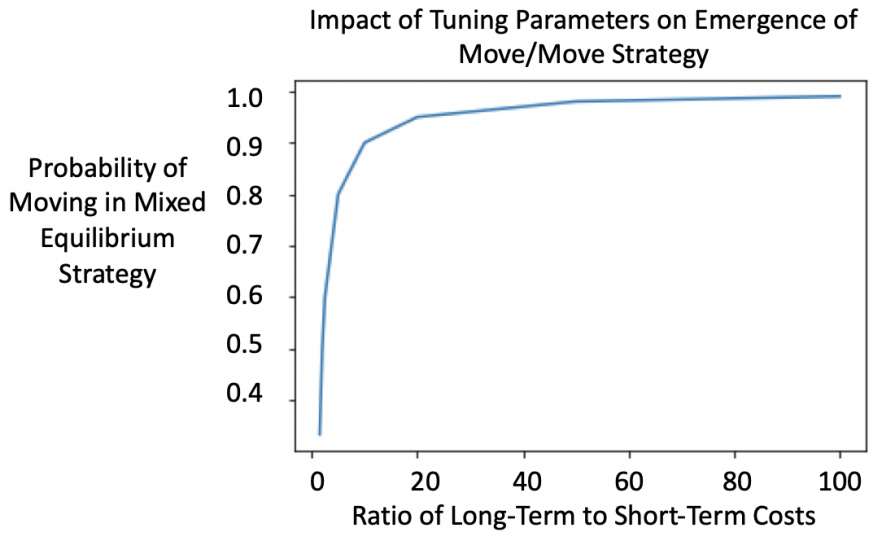


Figure 5.2: The impact of the ratio of long to short-term cost (f/g) on the emergence of a mixed-Nash equilibrium strategy where both satellites move with probability p, as shown on the y-axis.

Ratio (H/G)	**Probability (%)**
1.5	33
2	50
4.8	75
10	90
20	95

Table 5.3: The ratio of long to short-term cost (f/g) and its subsequent probability of a move-move strategy for both satellites.

never results in a pure strategy where both satellites maneuver, when the long-term cost exceeds the short-term cost by a factor of approximately 100, the resulting probability distribution leads to movement over 99% of the time. To analyze this trade-off in behavior, Table 5.3 shows the specific ratio for several probability thresholds of the move-move strategy. For all subsequent experiments, we will use ratio $(f/G) = 2$.

5.5 Repeated Game Methodology

We transform the one-shot game into a sequence of n repeated games. In the repeated game formulation, the decision shifts from a binary variable to maneuver to a discrete variable of when to maneuver, if at all. This requires each satellite to maintain a high-fidelity estimation of the satellite's current location and its best estimate of its adversary's location. We incorporate Kalman filtering to model the uncertainty estimates of both satellites over the

course of the decision period.

Figure 5.3 demonstrates the architecture for our repeated game. We elaborate on the architecture in greater detail in the following sections.

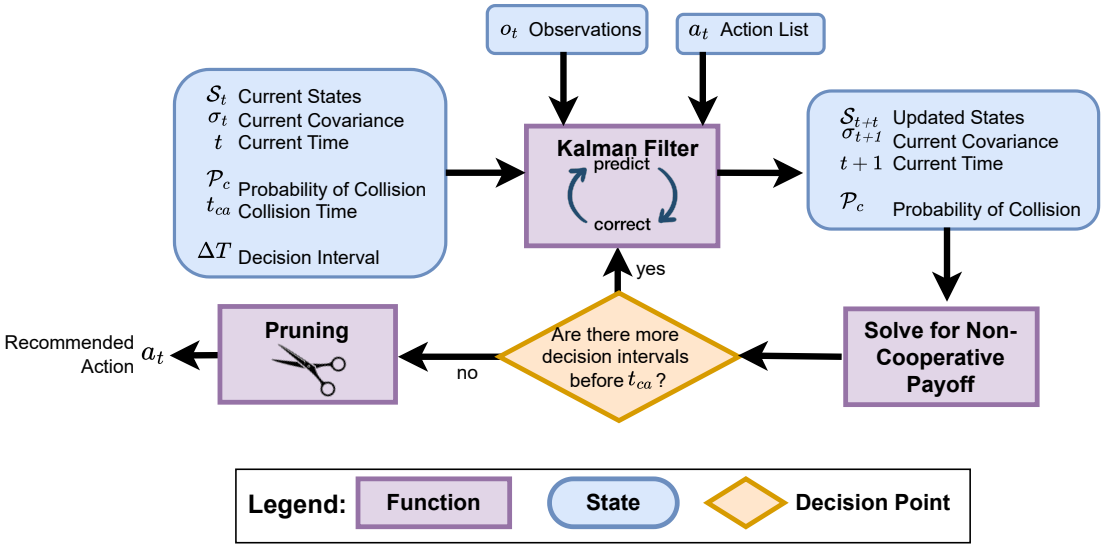


Figure 5.3: Overview of our Repeated n Step Game. **Kalman Filter**. We rely on a Kalman Filter for state estimation and covariance propagation over the course of the n repeated games. **Solve for Payoff**. Using the covariance estimates, probability of collision, and updated state estimation, we produce the payoff available to each satellite during that round. **Pruning**. Once the payoff is calculated for the total number of n rounds, we perform pruning to identify the best solution of the set.

5.5.1 Kalman Filter

Modeling the covariance uncertainty is central to the decision to maneuver. As demonstrated in Figure 5.1, we found that the relationship between satellite risk threshold and the probability of collision during each game determines satellite behavior. As a result, for the n step repeated game, we incorporate an Extended Kalman Filter to model the evolution of covariance propagation throughout the decision period.

The Kalman Filter consists of a linear, discrete-time, and time-varying system characterized by a finite-dimensional state vector and a sequence of noisy observations from which the state is inferred by minimizing a quadratic function of the estimation [120]. It involves a two-step procedure: predict and update. In the predict step, the state model is used to predict the current mean state estimate and covariance based on the knowledge up to the previous instant t_{k-1}. The *apriori* mean and covariance are defined as

$$\begin{aligned} \hat{x}_k^- &= \mathbb{E}[x_k | y_{1:k-1}] \\ P_k^- &= \mathbb{E}[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T | y_{1:k-1}] \end{aligned} \tag{5.3}$$

The superscript "-" is used to indicate predicted estimates. In the update step, the measurement is used to correct the predicted state producing the a posteriori distribution, denoted with the superscript $+$.

$$\begin{aligned} \hat{x}_k^+ &= \mathbb{E}[x_k|y_{1:k}] \\ P_k^+ &= \mathbb{E}[(x_k - \hat{x}_k^+)(x_k - \hat{x}_k^+)^T|y_{1:k-1}] \end{aligned} \tag{5.4}$$

To extend the applicability of the Kalman Filter to nonlinear systems like astrodynamics, we employ an Extended Kalman Filter (EKF). The system is approximated by a linearized version around the state estimate (using first-order Taylor series expansion), and then the conventional KF algorithm is applied. We provide additional information about the modeling of the predict and update steps in Sections 5.5.1 and 5.5.1.

Predict

In the predict step, the the mean and state transition matrix dynamics are propagated from time interval $t \in [t_{k-1}, t_k]$ from the initial conditions $\vec{x}(t_{k-1}) = \vec{x}_{k-1}^+$ and $\Phi(t_{k-1}, t_{k-1}) = I_{n\times n}$ with

$$\begin{aligned} \dot{\hat{x}}(t) &= f(\hat{x}(t), t) \\ \dot{\Phi}(t, t_{k-1}) &= F(\hat{x}(t), t)\Phi(t, t_{k-1}) \end{aligned} \tag{5.5}$$

Through integration $\hat{x}_k^- = \hat{x}(t)|_{t=t_k}$ and $\Phi(t_k, t_{k-1}) = \Phi(t, t_{k-1})|_{t=t_k}$

Then, we discretize the process noise covariance and compute the propagated state covariance.

$$Q_d(t_k, t_{k-1}) = \Phi(t_k, t_{k-1})Q\Phi^T(t_k, t_{k-1}) \cdot (t_k - t_{k-1}) \tag{5.6}$$

$$P_k^- = \Phi(t_k, t_{k-1})P_{k-1}\Phi^T(t_k, t_{k-1}) + Q_d(t_k, t_{k-1}) \tag{5.7}$$

where system Jacobian $F(\hat{x}(t), t)$ is

$$F(\hat{x}(t), t) = \left.\frac{\delta f(x(t), t)}{\delta x(t)}\right|_{x(t)=\hat{x}(t)} \tag{5.8}$$

Update

In the update step, the Kalman gain is computed, and the state mean and covariance are updated.

$$K_k = P_k^- H^T(\hat{x}_k^-, t_k)[H(\hat{x}_k^-, t_k)P_k^- H^T(\hat{x}_k^-, t_k) + R_k]^{-1} \tag{5.9}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(y_k - h_d(\hat{x}_k^-, t_k)) \tag{5.10}$$

$$P_k^+ = [I_{n\times n} - K_k H(\hat{x}_k^-, t_k)]P_k^- \tag{5.11}$$

where system Jacobian $H(\hat{x}_k^-, t_k)$ is

$$H(\hat{x}_k^-, t_k) = \left.\frac{\delta h_d(x_k, t_k)}{\delta x_k}\right|_{x_k=\hat{x}_k} \tag{5.12}$$

The process noise $w(t)$ is assumed to have constant covariance $Q(t) = Q$, which implies that the stochastic process has infinite variance. This is used to model system uncertainties and disturbances.

Propagation

For propagation between states within the Kalman Filter, we rely on Cowell's formulation to account for these additional perturbation accelerations to improve the state estimation accuracy within our model. As shown in Figure 5.4, Kepler's equation is sufficient for cases where a rough approximation of satellite motion is needed. However, additional perturbations can have a significant impact on the acceleration of a satellite.

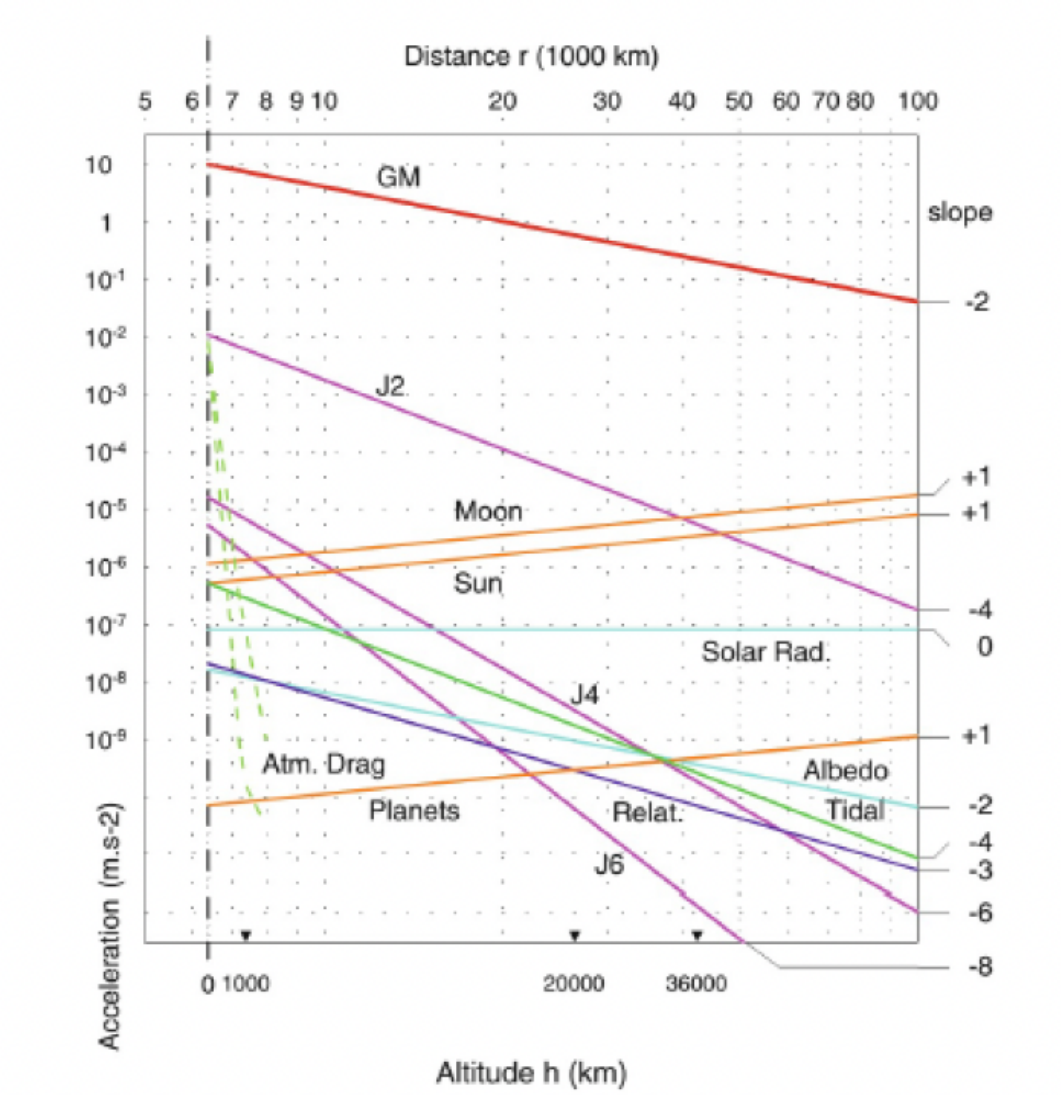


Figure 5.4: Magnitude of perturbing accelerations of Earth satellites as a function of their altitude. Figure from [1].

First we start with Cowell's formulation, which uses an additional term a_d to account for any disturbances.

$$\ddot{r} = -\frac{\mu}{||r||^3} + a_d \tag{5.13}$$

We model Earth's gravitational field and atmospheric drag, transforming Equation 5.13

into

$$\ddot{r} = -\frac{\mu}{||r||^3} + a_{\text{atmo}} + a_{\text{grav}} \tag{5.14}$$

The equation for the atmospheric drag is:

$$a_{\text{atmo}} = -\frac{1}{2}\frac{C_d A}{m}\rho||v_{rel}||v_{rel} \tag{5.15}$$

where the coefficient of drag C_D is a dimensionless quantity that reflects the satellite's susceptibility to drag forces. A and m are the cross-sectional area and mass of the satellite and ρ is the atmospheric density. v_{rel} is the velocity of the satellite relative to the local surrounding atmosphere. A reasonable approximation is to assume the atmosphere co-rotates with the Earth, resulting in $v_{rel} = v_i - w_{\oplus} \times r_i$.

We rely on the exponential model to model the density of the atmosphere but note that more complex models exist that incorporate solar and magnetic field activity ([121, 122]). The exponential model is

$$\rho = \rho_0 \exp(-\frac{h_{ellp} - h_0}{H}) \tag{5.16}$$

where the coefficients ρ_0, h_0, and H are a function of the satellite's altitude, h_{ellp}.

To determine the acceleration due to Earth's gravitational potential, we account for Earth's non-uniform mass distribution and non-spherical shape and symmetry by using a spherical harmonic series.

$$U = \frac{\mu_{\oplus}}{r}\left[1+\sum_{n=2}^{N_n} C_{n,0}\big(\frac{a_e}{r}\big)^n P_{n,0}\sin\phi_{gc}+\sum_{n=2}^{N_n}\sum_{m=1}^{\min(n,N_m)}\big(\frac{a_e}{r}\big)P_{n,m}\sin\phi_{gc}[C_{n,m}\cos(m\lambda)+S_n, m\sin(m\lambda)]\right] \tag{5.17}$$

where U is expressed as a function of the satellite's Earth Centered Earth Fixed geocentric coordinates (r, ϕ_{gc}, λ), where r is the radial distance, ϕ_{gc} is the geocentric latitude, and λ is the longitude. Each harmonic term in the series is specified by its degree and order (n, m). The associated gravitational acceleration can be given by

$$a_{\text{grav}} = \frac{\delta U}{\delta r}\left(\frac{\delta r}{\delta r_f}\right)^T + \frac{\delta U}{\delta \phi_{gc}}\left(\frac{\delta \phi_{gc}}{\delta r_f}\right)^T + \frac{\delta U}{\delta \lambda}\left(\frac{\delta \lambda}{\delta r_f}\right)^T \tag{5.18}$$

This makes the acceleration vector $a_{\text{grav}} = [a_{xf}, a_{yf}, a_{zf}]^T$

$$a_{xf} = \left[\frac{1}{r}\frac{\delta U}{\delta r} - \frac{z_f}{r^2\sqrt{x_f^2+y_f^2}}\frac{\delta U}{\delta \phi_{gc}}\right]x_f - \left[\frac{1}{x_f^2+y_f^2}\frac{\delta U}{\delta \lambda}\right]y_f - \frac{\mu_{\oplus} x_f}{r^3} \tag{5.19}$$

$$a_{yf} = \left[\frac{1}{r}\frac{\delta U}{\delta r} - \frac{z_f}{r^2\sqrt{x_f^2+y_f^2}}\frac{\delta U}{\delta \phi_{gc}}\right]y_f - \left[\frac{1}{x_f^2+y_f^2}\frac{\delta U}{\delta \lambda}\right]x_f - \frac{\mu_{\oplus} y_f}{r^3} \tag{5.20}$$

$$a_{zf} = \frac{1}{r}\frac{\delta U}{\delta r}z_f + \frac{\sqrt{x_f^2+y_f^2}}{r^2}\frac{\delta U}{\delta \phi_{gc}} - \frac{\mu_{\oplus} z_f}{r^3} \tag{5.21}$$

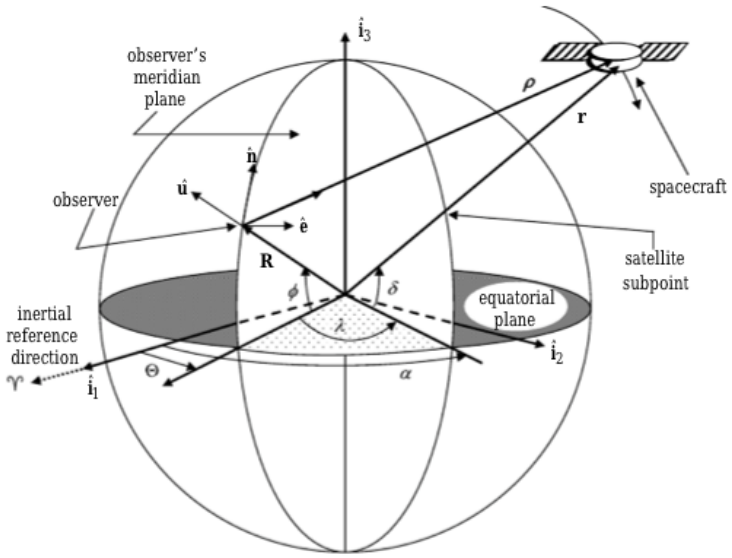


Figure 5.5: Geometry of Earth Observations of Spacecraft Motion. Figure taken from [2]

5.5.2 Measurement Update

We assume an earth-based observer (i.e., a radar site) that measures the range, azimuth, and elevation of a spacecraft in orbit. The geometry of the observer relative to the satellite is shown in Figure 5.5. ϱ is the slant range, $\mathbf{r}$ is the radius vector locating the spacecraft, $\mathbf{R}$ is the radius vector locating the observer, α and δ are the right ascension and declination of the spacecraft, respectively, Θ is the sidereal time of the observer, ϕ_{gc} is the latitude of the observer, and λ is the east longitude from the observer to the spacecraft. The observation is given by:

$$\varrho = \mathbf{r} - \mathbf{R} \tag{5.22}$$

In the non-rotating equatorial (inertial) components, the vector ϱ is given by:

$$\varrho = \begin{bmatrix} x - ||\mathbf{R}|| \cos(\phi_{gc}) \cos(\Theta) \\ y - ||\mathbf{R}|| \cos(\phi_{gc}) \sin(\Theta) \\ z - ||\mathbf{R}|| \sin(\phi_{gc}) \end{bmatrix} \tag{5.23}$$

where x, y, and z are the components of the vector $\mathbf{r}$. The conversion from the inertial to the observer coordinate system is given by:

$$\begin{bmatrix} \varrho_u \\ \varrho_e \\ \varrho_n \end{bmatrix} = \begin{bmatrix} \cos(\phi_{gc}) & 0 & \sin(\phi_{gc}) \\ 0 & 1 & 0 \\ -\sin(\phi_{gc}) & 0 & \cos(\phi_{gc}) \end{bmatrix} \begin{bmatrix} \cos(\Theta) & \sin(\Theta) & 0 \\ -\sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.24}$$

Next, we define a radar sight that measures the azimuth, az, elevation el, and range, ρ. The observation equations are given by

$$||\varrho|| = (\varrho_u^2 + \varrho_e^2 + \varrho_n^2)^{1/2} \tag{5.25}$$

$$az = \tan^{-1}\left(\frac{\varrho_e}{\varrho_n}\right) \tag{5.26}$$

$$el = \sin^{-1}\left(\frac{\varrho_u}{||\varrho||}\right) \tag{5.27}$$

Measurement Noise

Measurement observations are imperfect and corrupted by noise that is inevitable in real-world sensors. The measurement noise sequence $v(t_k)$ quantifies the random uncertainty associated with the observation error. It is assumed to be a sequence of white, zero-mean Gaussian noise with covariance R_k at time t_k. It is reasonable to assume that all ground stations produce observations that are independent of one another. The covariance matrix is expressed by

$$R_k = \text{diag}(R_k^1, ..., R_k^r) \tag{5.28}$$

where $\text{diag}(\cdot)$ is the diagonal matrix formed from the input parameters r is the number of available stations at t_k and R_k^j is the covariance associated to station j which incorporates the standard deviations of the noise sequences associated to each observation in the set $\{\varrho^j, az^j, el^j, \dot{\varrho}^j$, according $R_k^j = \text{diag}((\sigma_\varrho^j)^2, (\sigma_a^j z)^2, (\sigma_e^j l)^2, (\sigma_{\dot{\varrho}}^j)^2)$, where it is assumed that each station produces independent uni-dimensional observations.

Ground Station Availability

Observation vector $y(t_k)$ is formed from measurements of all available ground stations. Therefore a condition for station availability must be defined. A ground station is said to be unavailable when the line of sight path between the target satellite and the observer ground station is blocked by the Earth. An availability (field of view) condition can be defined by assuming that the ground station tracking capabilities are limited to some azimuth and elevation angular intervals. Therefore, measurements are only available when $az \in [az_{min}, az_{max}]$ and $el \in [el_{min}, el_{max}]$. We assume that viewing the entire azimuth domain is possible, but only positive elevation angles (i.e. above the horizontal) plane are deemed as available.

For all experiments, we define a single ground station is located at the Haystack Observatory in Westford, Massachusetts longitude $\lambda = -71.44°$, latitude $\phi_{gd} = 42.58°$. The sensor measurement noise deviations are shown in Table 5.4. These noise statistics were chosen as they represent typical values of LEO radar tracking systems [99, 123].

Measurement Type	**Numerical Value**
Range	100 m
Azimuth and Elevation	0.02 °
Range Rate	10 cm s^{-1}

Table 5.4: Characteristic properties of the satellites in our simulation

5.5.3 Generating Payoff Matrix

The payoff matrix in the repeated game has several adaptions from the single-game case. First, the cost to each satellite associated with a decision at time $t \in [1, T]$ varies depending

on the number of decision set points left and the probability of collision.

We adapt the cost to maneuver to the repeated game case.

$$g_i^t(a_i^t) = G^t a_i^t = \frac{G^{\max}}{k^{T-1-t}} a_i^t \; \forall t = \{1, \ldots, T-1\} \tag{5.29}$$

for some $G^{\max} > 0, k \geq 1$. If a satellite does not maneuver and $a_i^t = 0$, the cost to its satellite is 0. If the satellite does maneuver, the satellite incurs a cost of G^t, with maximum value $G^{\max}$. For the repeated game, we've adapted the cost to increase with time, as suggested by the power law term k^{T-1-t} because a decreased time to the closest approach requires more thrust to get a corresponding change in the DCA.

$$u_i^T = \sum_{t=1}^{T} G_i^t + H_i^t \tag{5.30}$$

We employ a similar piecewise structure to model the terminal cost as detailed in Section 5.3. We make one modification. The satellite's risk threshold $\theta_i[t]$ is a function of the number of rounds left in the game.

$$\theta_i[t] = -\theta \exp^{T-t} \tag{5.31}$$

With this adjustment, as the time to collision draws closer, the satellites will naturally become more risk averse. Consequently, their risk tolerance decreases. This type of variable is independent of and known to the other satellite.

Probability of Collision

For the repeated game case, the probability of collision can evolve over time as the covariance of the two respective satellites fluctuates. To calculate the probability of collision at each time-step, we rely on the Foster and Estes 2D-P_C method to quantify collision risks [119]. The probability of collision problem is transformed into a 2D collision plane by assuming the rectilinear motion of the two objects during the encounter plane. This plane is defined as perpendicular to the relative velocities of the two objects and assumes that the combined uncertainty along the relative velocity vector has no bearing on the calculation of the probability of collision.

Assuming a circular cross-sectional area as the hard body radius (HBR), the resultant probability of collision is:

$$P_c = \frac{1}{2\pi\sqrt{|C|}} \int_{-HBR}^{HBR} \int_{-\sqrt{HBR^2-x^2}}^{\sqrt{HBR^2-x^2}} \exp\left(-\frac{1}{2}(\vec{r} - \vec{r_d})^T C^{-1} (\vec{r} - \vec{r_d})\right) d\vec{z}; d\vec{x} \tag{5.32}$$

C is the combined position covariance for both objects, $\vec{r}$ is the position on the collision plane, and $\vec{r_d}$ is the debris object's position on the conjunction plane. These positions are given by

$$\vec{r} = \begin{bmatrix} x \\ z \end{bmatrix} \tag{5.33}$$

$$\vec{r_d} = \begin{bmatrix} x_1 - x_2 \\ z_1 - z_2 \end{bmatrix}. \tag{5.34}$$

5.5.4 Optimization and Solution

At each round of the game, the probability of collision for each strategy is calculated. The risk threshold of the controlled satellite and the covariance of the system are also updated. These two updates create unique short-term and long-term costs associated with that state for each of the four strategic outcomes (move-move, move-wait, wait-move, and wait-wait); these costs are saved for each round. To create the decision tree, the costs at each round are added together.

As we are interested in the optimal time to maneuver given a set of n rounds, we constrain the tree so that the satellite can only maneuver once throughout the game.

5.6 Validation

We validate our propagation model against real-world covariance measurements from the MIT ARCLab Prize for AI Innovation in Space Challenge [124]. We assume the following characteristic properties of the satellite for our model both for validation and in the presented results unless mentioned otherwise.

Satellite Property	**Numerical Value**
Mass m	25 kg
Drag Area A_{drag}	0.5 m^2
Coefficient of Drag C_D	2.0

Table 5.5: Characteristic properties of the satellites in our simulation

Tables 5.6 and 5.7 compares the true measured data from the Prize for AI Innovation Challenge with the EKF position estimate. The error is the difference between the two position values, normalized by the true position. We compared the performance of our algorithm across 10 hours within one day and found small error rates (<1%).

Time	**Δ T**	**True Position**	**EKF Position**
9/1/2022 03:58:19	2:19:20.38	[4525.68,-4587.3, 3285.21]	[4526.57, -4586.40, 3285.64]
9/1/2022 06:18:31	2:20:12.32	[6155.45, -1638.9, 4259.09]	[6157.30, -1633.35, 4259.95]
9/1/2022 08:14:36	1:56:05.06	[1334.04,-6780.89,1262.79]	[1336.49, -6779.93, 1264.38]
9/1/2022 11:11:06	2:56:30.05	[4097.36, 8339.84, 2298.73]	[4108.08, 8330.28 , 2306.46]

Table 5.6: Position Differences between the **True Position** from the Prize for AI Innovation Challenge, and the **EKF Position** estimate

Time	Error (%)
9/1/2022 03:58:19	0.08
9/1/2022 06:18:31	0.064
9/1/2022 08:14:36	0.04
9/1/2022 11:11:06	0.17

Table 5.7: Error between the **True Position** from the Prize for AI Innovation Challenge, and the **EKF Position** estimate

5.7 Results

Table 5.8: Recommended Strategies for Different N-step games

Maximum Iterations in Repeated Game	Time Step Interval	Recommended Strategy
2	3 hours	Force opponent to maneuver
3	2 hours	Move at first instance
4	1.5 hours	Maneuver at time steps 2 and 3

To determine the impact of the number of steps in the game on the resultant strategies between satellites, we compare strategies between different iterations of n-step games. Table 5.8 provides an overview of the recommended strategies for various numbers of iterations, along with the corresponding decision-making intervals for each time step. Our findings reveal that as the number of time steps increases, satellites are more inclined to execute maneuvers earlier in the game. For example, in a scenario with four iterations, we observe multiple solutions: one strategy recommends initiating a maneuver at time step 2, while another suggests maneuvering at time step 3.

These findings highlight a key trend: as the number of decision-making intervals grows, satellites are increasingly likely to adopt proactive avoidance strategies. This behavior can be attributed to the dynamic nature of the risk thresholds associated with each satellite. Specifically, the risk threshold diminishes progressively with each time step, meaning that a collision risk deemed acceptable at an earlier stage might exceed the tolerable threshold at a later time. Consequently, satellites are incentivized to act preemptively to mitigate potential collisions, reflecting a shift towards early and deliberate maneuvering as the game progresses.

The computational complexity of this approach poses significant challenges. With each branching decision, the number of possible states increases exponentially, making the computational cost of evaluating later iterations (e.g. n=30) significantly higher. This complexity impacts the feasibility of scaling the method to larger problems or more frequent decision-making intervals. However, despite the increased computational burden, the results provide critical insights: satellites are more likely to adopt preemptive avoidance strategies as the number of decision-making intervals increases. This behavior arises because the dynamic

risk thresholds for each satellite decrease over time, making previously acceptable collision risks intolerable at later stages.

Ultimately, while the branched tree search method is computationally intensive, it enables a detailed exploration of the strategic space, ensuring that the resultant strategies are robust and well-suited to the dynamic nature of the game. This underscores the importance of balancing computational resources with the need for accurate and actionable decision-making in satellite operations.

5.8 Chapter summary and contributions

This chapter has discussed the development of a game theoretic framework to model competitive satellite collision avoidance. The initial part of the chapter explained the results from the one-step game. Then, we extended the one-step case to a repeated game formulation with more complex estimation and risk threshold modeling in Section 5.5.

The specific contributions of this chapter are as follows:

Contribution 5.1	One-shot game framework to model decision-making for non-cooperative satellite collision avoidance.
Contribution 5.2	Examination of the relationship between different satellite characteristics and conjunction parameters to study the emergence of strategic behaviors in satellite collision avoidance.
Contribution 5.3	Formulation of an n-step game theoretic formulation for satellite collision avoidance problem.

Chapter 6

Conclusion

This chapter summarizes the contributions of this dissertation. We expand on various discussion points and suggest avenues for future work.

6.1 Summary of Contributions

The thesis statement of this work is that selective communication between satellites can have significant advantages for decentralized space traffic management.

In previous chapters, we found that the type of encoding used for communication between agents significantly influences the behavior that satellites learn. We developed a simulation environment for satellite proximity operations to assess the impact of various reinforcement learning approaches. First, we investigated the effect of learned information sharing in decentralized settings and compared its performance to centralized algorithms with perfect information sharing. Our empirical results show that our algorithm achieves near-optimal performance, even compared to centralized methods with access to much more information. We also demonstrate how transfer learning can accelerate training by initially training a ground-based model for a short time and then transferring the resulting weights to the space environment. Furthermore, we show that our algorithm generalizes well to different tasks and environmental perturbations, consistently exhibiting goal-reaching behaviors across varied environments and navigation tasks. Using our graph formulation, we evaluate the significance of sharing the intended goal location and find that goal sharing improves path planning. These findings offer initial insights into the foundations of learning-based controllers for collision avoidance in space.

We then relax the assumption that all communication and actions between agents must occur synchronously. We introduce AsyncCoMARL, a multi-agent reinforcement learning framework designed for coordination in environments where agent actions and communications occur at asynchronous intervals. Our results show that graph transformers are an effective encoder in this setting, outperforming other graph-based attention mechanisms. We also highlight the effectiveness of independent actor-critic formulations and demonstrate that our algorithm offers greater scalability than other asynchronous techniques.

Our final contribution focuses on understanding collision avoidance in non-cooperative scenarios. We formalize a single-step payoff matrix based on the satellite's risk threshold,

maneuver costs, and the estimated probability of collision. By evaluating the influence of weighting on the risk threshold and maneuver cost variables, we identify the emergence of a mixed Nash equilibrium strategy where both satellites maneuver when the cost-to-risk ratio is significantly higher. We then extend our analysis to multi-step collision avoidance, where satellites must determine the best time to maneuver over a series of n repeated games. In this repeated game formulation, we find that the optimal maneuver time is earlier rather than closer to the collision event.

This work contains contributions in multi-agent systems, satellite control, space traffic management, MARL, and Game Theory; however, we believe the largest contribution of this dissertation is highlighting the feasibility of a decentralized coordination control scheme for collision avoidance in space. In future developmental work of multi-agent systems in space, we hope our work encourages researchers and engineers to consider decentralized learning structures when designing and implementing systems with multiple individual satellites.

6.2 Implications for Space Traffic Management

This thesis was initially motivated by the problem of space traffic management and how to develop tools and frameworks that can advance autonomy in this domain. The balance between centralization and decentralization in space traffic management presents a compelling research challenge, as it requires navigating trade-offs between global coordination and localized autonomy. While autonomous centralized, global methods offer significant computational power and a comprehensive overview of the system, they may be less practical or resilient in space's dynamic and uncertain environment. Centralized systems depend on communication with a central authority, which introduces latency, single points of failure, and vulnerability to communication disruptions. In contrast, distributed, local approaches leverage onboard autonomy to enable faster decision-making tailored to the specific context of each satellite. This decentralization enhances system resilience, scalability, and adaptability, particularly in scenarios with limited or delayed communication, which can occur in space operations. Additionally, local approaches align more naturally with the constraints of orbital mechanics and the autonomy required for multi-satellite constellations, reducing dependency on real-time global coordination while maintaining effective collision avoidance and strategic maneuvering.

Multi-agent reinforcement learning offers a promising framework to address the challenges posed by the current orbital environment. In a congested orbital climate, MARL can enable satellites to learn cooperative strategies for collision avoidance and efficient use of limited orbital slots, reducing risks and optimizing shared resources. By continuously adapting to the dynamic and uncertain nature of the orbital environment, this thesis presented an initial investigation into MARL's feasibility in navigating complex interactions effectively, ensuring safer and more efficient operations.

Addressing the autonomy stack is one piece, but space traffic management is particularly challenging due to the heterogeneity of satellites, each with differing levels of technological capability and operational expertise, complicating coordination and the implementation of effective strategies. In this thesis, we have developed tools to model strategic decision-making between competitive and cooperative satellites, but further research is needed to scale these

methods effectively to represent the complex, mixed environments of diverse space actors.

For example, constellation operators increasingly control a significant portion of active satellites, yet we neglected to consider constellation interactions in this work. Given the scale of constellations, which can range from tens to thousands of satellites, some degree of centralization could facilitate more efficient decision-making and coordination when engaging with this type of distributed space system. Therefore, a hybrid model incorporating centralized oversight for certain operational aspects while allowing for local autonomy and decentralized decision-making could offer a balanced solution, catering to cooperation and competition while maintaining robust space traffic management.

6.3 Future Work

There are several areas for future research directions inspired by the work in this dissertation. Some directions are direct short-term extensions of the work presented in the previous chapters, while others are broader long-term expansions.

6.3.1 Short-Term Extensions

Initial directions for future research include expanding to different dynamical models, like Gauss variational equations, incorporating agents with different maneuvering abilities, and implementing hierarchical control methodologies where the reinforcement learning algorithm is used as a high-level planner and optimal control techniques are used for lower-level satellite control.

Our work in Chapter 3 emphasizes how transfer learning from ground-based environments can improve training efficiency for multi-agent reinforcement learning in the space domain. We note that the ground-based network takes comparatively few training steps to learn goal-reaching behavior. This raises an important question about the nature of transfer learning in this domain and whether additional learning methodologies like zero-shot learning or few-shot learning can be applied to recognize collision avoidance in different dynamic scenarios.

In Chapter 3, we also relied on the graph structure to determine the significance of maneuver-sharing information on performance. One natural extension of this work is to explore the significance of other commonly withheld pieces of information, such as the attitude or ballistic coefficient. This work could be used to more directly inform policy measures for space traffic management (i.e., what information should satellites be required to share and what the consequences of not sharing).

Further developing the game theoretic collision avoidance framework is another interesting area of future research. We have presented an initial prototype for a system where the risk threshold is modeled as a continuous exponential function throughout the simulation. Our preliminary evaluation shows that satellites in the repeated-step collision avoidance scenario can produce higher payoffs when maneuvering sooner in the decision period. However, our prototype remains simplistic. Further work developing a more complex risk model that can account for mission lifetime, upcoming planned maneuvers, impact on scientific return, and operational costs is an exciting area of future work that could inform satellite policy decision-making in this arena. With the incorporation of a more advanced risk model, the

game theoretic tool could be used to examine more nuanced trade-offs common in satellite decision-making for collision avoidance. This work will require addressing broader problems of non-convex optimization and covariance propagation and further developing and understanding the connections between different design variables.

6.3.2 Broader Long-Term Extensions

Recall from Chapter 4 where we demonstrated that agents can work together asynchronously, meaning each agent acts and communicates at some interval κ. One natural extension of this formulation is to examine the relationship between communication and emergent teamwork behaviors among subsets of agents. The ability to effectively work in teams can magnify a group's abilities beyond the capabilities of any one individual. However, the impact of team structure on multi-agent learning is still poorly understood. The idea that teamwork behaviors can naturally emerge from certain classes of information, or frequency of information, is an interesting concept for multi-agent learning. For example, in settings where a singular agent observes something scientifically interesting, the ability to synchronize communication with others and incentivize collaboration with subsets of nearby agents would be useful for science exploration. Fully exploring the relationships between subsets of agents in teams and their communications is an interesting area for future work.

Similarly, it would be interesting to explore interactions between individual satellites and large constellation networks, particularly in terms of how smaller operators can coexist with or compete against larger, more centralized systems. Understanding these dynamics could reveal opportunities for improved coordination, resource sharing, and collision avoidance, ensuring that both small-scale and large-scale operations can coexist safely and efficiently in a congested orbital environment.

Finally, the safety of multi-agent reinforcement learning in space operations is a critical concern. Since MARL involves agents that learn through interaction and reward-based feedback, there is a risk that agents might learn unsafe behaviors, such as aggressive maneuvers, collisions, or other unintended consequences, especially in environments with limited supervision or oversight. To address these concerns, safety mechanisms must be integrated into the learning process, such as defining explicit safety constraints in the reward function or using techniques from control theory like control barrier functions. For MARL to be deployed as a controller or decision-making tool in an operational space flight system, it will be essential to demonstrate that its behavior and safety meet or exceed the standards of existing techniques.

Appendix A

SpaceMARL Implementation and Supplemental Experiments

This section provides additional discussion surrounding the implementation and performance of algorithms discussed in Chapter 3.

A.1 Baseline Implementation Sources

We modified the codebases from the official implementations for the GPG, DGN, EMP, and MAPPO baselines and a thoroughly benchmarked codebase for MADDPG, MATD3, VDN and QMIX and provide the links to those implementations here. Note that we used the same hyperparameters as used in their original implementations assuming that they were optimal. We performed a hyperparameter search for these algorithms by varying the learning-rates, network size and a few algorithm specific parameters but did not find a better set of hyperparameters for the environment and chose to report with the original hyperparameters.

1. MAPPO https://github.com/marlbenchmark/on-policy
2. MADDPG, MATD3, QMIX, VDN https://github.com/marlbenchmark/off-policy

Table A.1 and A.2 show key learning parameters.

Table A.1: Hyperparameters used in Space-MARL

hyperparameters	Value
entity embedding layer dim	3
entity hidden dim	16
num embedding layer	1
add self loop	False
gnn layer hidden dim	16
num gnn heads	3
num gnn layers	2
gnn activation	ReLU

Table A.2: Common Hyperparameters used in MADDPG, MATD3, QMIX, VDN

Common Hyperparameters	Value
gradient clip norm	10.0
random episodes	5
epsilon	$1.0 \rightarrow 0.05$
epsilon anneal time	50000 timesteps
train interval	1 episode
gamma	0.99
critic loss	mse loss
buffer size	5000 episodes
batch size	32 episodes
optimizer	Adam
optimizer eps	1e-5
weight decay	0
network initialisation	Orthogonal
use reward normalisation	True
use feature normalisation	True

A.2 Performance with Agents Involving Heterogeneous Noise

We perform a secondary experiment in which two noisy satellites with differing σ values are interacting σ_1 and σ_2. Both sigma values are set such that they are above the minimum sigma threshold for effective path-planning σ_{min}.

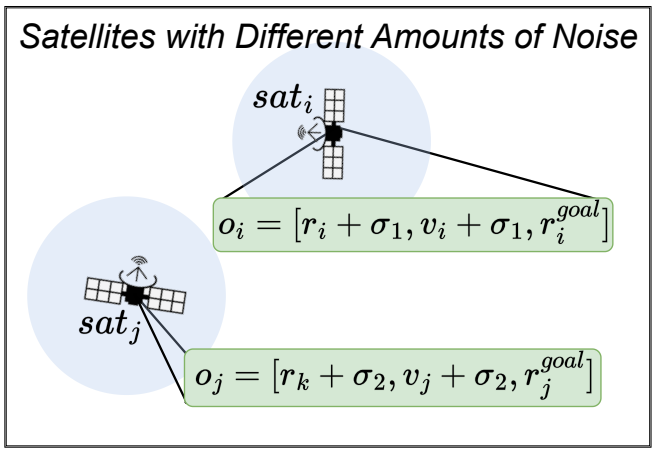


Figure A.1: Experiment involving two satellites with different amounts of noise in their observation

We use 100 episodes for evaluation. We rely on several metrics to characterize performance. First, we use the distance traveled by the agent type per episode to assess the path planning abilities of the agent. Next, we use the success rate to determine its goal-reaching ability. Finally, we show the average reward per episode to assess InfoMARL's performance overall.

	Satellite Noise σ_1	Satellite Noise σ_2
Success Rate	37	10
Control Action per Episode	23	24

Total Episode Statistics	
Average Reward per Episode	-14.31
Number Col per Episode	0.28
Success Rate	4

Table A.3: Performance of two satellite noise experiment across 100 evaluation episodes in *navigation* task.

As Table A.3 demonstrates, SpaceMARL struggles to differentiate between the two agents and performs poorly as a result. The individual agents can reach their goals at a higher rate than the total episode success rate (37 vs 4, 10 vs 4), indicating that the algorithm can learn individual goal-reaching but not more nuanced coordination and differentiation. The higher success rate of agent 1 is likely due to the fact that its noise (σ_1) is smaller than agent 2 (σ_2), and thus its true state is less obscured.

Appendix B

AsyncCoMARL Implementation and Supplemental Experiments

This section provides additional discussion surrounding the implementation and performance of algorithms discussed in Chapter 4.

B.1 Baseline Implementation Sources

We modified the codebases from the official implementation from async-MAPPO, CACOM, DGN, and GCS baselines. Note that we used the same hyper parameters as used in their original implementations assuming that they were optimal.

1. async-MAPPO https://github.com/yang-xy20/async_mappo/tree/main
2. CACOM: https://github.com/LXXXXR/CACOM/tree/main
3. DGN: https://github.com/jiechuanjiang/pytorchDGN
4. GCS: https://github.com/LXXXXR/GCS_aamas337/tree/master

B.2 Key Dependencies

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

- gym = 0.26.2
- torch = 1.13.1
- torch-geometric = 2.3.1
- tensorboardX = 2.6.2.2
- wandb = 0.17.4

B.3 Hyperparameters

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 × buffer length × 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 B.1: Common Hyperparameters used in asyncMAPPO, GCS, and ACM

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

Table B.2: Common Hyperparameters used in GCS

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 B.3: Common Hyperparameters used in CACOM

Bibliography

[1] M. Capderou. *Handbook of Satellite Orbits: From Kepler to GPS*. Springer, 2014.

[2] J.L. Crassidis and J.L. Junkins. *Optimal Estimation of Dynamic Systems*. Chapman & Hall/CRC Applied Mathematics & Nonlinear Science. CRC Press, 2004.

[3] Daniel Jang. *Modeling the Future Space Debris Population and Orbital Capacity*. PhD thesis, Massachusetts Institute of Technology, 2024.

[4] Nicholas Johnson. The collision of iridium 33 and cosmos 2251: The shape of things to comeg. *International Astronautical Congress*, October 2009.

[5] Phillip Anz-Meador, John Opiela, and Jer-Chyi Liou. *History of On-Orbit Satellite Fragmentations*. National Aeronautics and Space Administration, 16th edition, 2022.

[6] Andrew Jones. Breakup of china's yunhai-1 (02) satellite linked to space debris collision. *SpaceNews*, Jan 2022.

[7] Carmen Pardini and Luciano Anselmo. The short-term effects of the cosmos 1408 fragmentation on neighboring inhabited space stations and large constellations. *Acta Astronautica*, 210:465–473, 2023.

[8] Jeff Foust. https://spacenews.com/spacex-and-oneweb-spar-over-satellite-close-approach/. *SpaceNews*, Apr 2021.

[9] Niko A. Grupen, Bart Selman, and Daniel D. Lee. Fairness for cooperative multi-agent learning with equivariant policies, 2021.

[10] The Consultative Committee for Space Data Systems. Recommendation for space data system standards conjunction data message. https://public.ccsds.org/Pubs/508x0b1e2s.pdf, June 2013.

[11] H. G. Lewis and G. Skelton. Space Debris Removal: A Game Theoretic Analysis. *2nd Orbital Debris Conference*, 7, 2023.

[12] James Rendleman and Sarah Mountin. Responsible ssa cooperation to mitigate on-orbit space debris risks. In *International Conference on Recent Advances in Space Technologies*, page 851–856, 2015.

[13] Charity Weeden, Jonathan Goff, Jessica Noble, Jeremy Schiel, Dana Turse, Colin Doughan, John Carrico, and Rob Patterson. Comments of global newspace operators. https://www.fcc.gov/ecfs/filing/1040578949828., Apr 2019.

[14] William Wiltshire, Patricia Cooper, and David Goldman. Comments of space exploration technologies corp. https://www.fcc.gov/ecfs/document/104050365604744/1, Apr 2019.

[15] Maureen McLaughlin. Comments of iridium communications inc. https://www.fcc.gov/ecfs/document/104050570315757/1, Apr 2019.

[16] Petra Vorwig and Suzanne Malloy. Comments of ses americom, inc. and o3b limite, 2019. https://www.fcc.gov/ecfs/filing/10405569822298, Apr 2019.

[17] Brian D Weimer, Douglas Svor, Samuel Swoyer, and Mariah Dodson Shuman. Comments of worldvu satellites limited. https://www.fcc.gov/ecfs/search/search-filings/filing/1040650203789, Apr 2019.

[18] Joseph C Anders. Comments of leosat ma, inc. https://www.fcc.gov/ecfs/search/search-filings/filing/104051333319010, Apr 2019.

[19] Richard S. Sutton and Andrew G. Barto. *Introduction to reinforcement learning*. MIT Press, 2 edition, 2012. (draft 2nd ed.).

[20] Brian Gaudet, Richard Linares, and Roberto Furfaro. Deep reinforcement learning for six degree-of-freedom planetary landing. *Advances in Space Research*, 65(7):1723–1741, 2020.

[21] Andrea Scorsoglio, Andrea D'Ambrosio, Luca Ghilardi, Brian Gaudet, Fabio Curti, and Roberto Furfaro. Image-based deep reinforcement meta-learning for autonomous lunar landing. *Journal of Spacecraft and Rockets*, 59(1):153–165, 2022.

[22] Callum Wilson and Annalisa Riccardi. Improving the efficiency of reinforcement learning for a spacecraft powered descent with q-learning. *Optimization and Engineering*, 2021.

[23] Lakshay Arora and Atri Dutta. Reinforcement learning for sequential low-thrust orbit raising problem. *AIAA Scitech 2020 Forum*, 2020.

[24] Andrew Harris, Thibauld Teil, and Hanspeter Schaub. *29th AAS/AIAA Space Flights Mechanics Meeting*. 2019.

[25] Harry Holt, Roberto Armellin, Nicola Baresi, Yoshi Hashida, Andrea Turconi, Andrea Scorsoglio, and Roberto Furfaro. Optimal q-laws via reinforcement learning with guaranteed stability. *Acta Astronautica*, 187:511–528, 2021.

[26] Daniel Miller, Jacob A Englander, and Richard Linares. Interplanetary low-thrust design using proximal policy optimization. In *2019 AAS/AIAA Astrodynamics Specialist Conference*, number GSFC-E-DAA-TN71225, 2019.

[27] J Elkins, R Sood, and C Rumpf. Autonomous spacecraft attitude control using deep reinforcement learning. In *71st International Astronautical Congress (IAC)*, volume 2020, 2020.

[28] James T Vedant. Reinforcement learning for spacecraft attitude control. In *70th International Astronautical Congress*, 2019.

[29] Hongyang Dong, Xiaowei Zhao, and Haoyang Yang. Reinforcement learning-based approximate optimal control for attitude reorientation under state constraints. *IEEE Transactions on Control Systems Technology*, 29(4):1664–1673, 2021.

[30] Charles E. Oestreich, Richard Linares, and Ravi Gondhalekar. Autonomous six-degree-of-freedom spacecraft docking with rotating targets via reinforcement learning. *Journal of Aerospace Information Systems*, page 1–12, 2021.

[31] Lorenzo Federici, Boris Benedikter, and Alessandro Zavoli. Deep learning techniques for autonomous spacecraft guidance during proximity operations. *Journal of Spacecraft and Rockets*, 58(6):1774–1785, 2021.

[32] Kirk Hovell and Steve Ulrich. Deep reinforcement learning for spacecraft proximity operations guidance. *Journal of Spacecraft and Rockets*, 58(2):254–264, 2021.

[33] Brenton Smith, Rasit Abay, Joshua Abbey, Sudantha Balage, Melrose Brown, and Russell Boyce. Propulsionless planar phasing of multiple satellites using deep reinforcement learning. *Advances in Space Research*, 67(11):3667–3682, 2021.

[34] Stefano Silvestrini and Michèle Lavagna. Neural-based predictive control for safe autonomous spacecraft relative maneuvers. *Journal of Guidance, Control, and Dynamics*, 44(12):2303–2310, 2021.

[35] Andrea Brandonisio, Michèle Lavagna, and Davide Guzzetti. Reinforcement learning for uncooperative space objects smart imaging path-planning. *The Journal of the Astronautical Sciences*, 68(4):1145–1169, Nov 2021.

[36] Yixin Huang, Zhongcheng Mu, Shufan Wu, Benjie Cui, and Yuxiao Duan. Revising the observation satellite scheduling problem based on deep reinforcement learning. *Remote Sensing*, 13(12):2377, 2021.

[37] Haijiao WANG, Zhen YANG, Wugen ZHOU, and Dalin LI. Online scheduling of image satellites based on neural networks and deep reinforcement learning. *Chinese Journal of Aeronautics*, 32(4):1011–1019, 2019.

[38] Xuexuan Zhao, Zhaokui Wang, and Gangtie Zheng. Two-phase neural combinatorial optimization with reinforcement learning for agile satellite scheduling. *Journal of Aerospace Information Systems*, 17(7):346–357, 2020.

[39] Massimo Tipaldi, Raffaele Iervolino, and Paolo Roberto Massenio. Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges. *Annual Reviews in Control*, 54:1–23, 2022.

[40] Brian Gaudet, Richard Linares, and Roberto Furfaro. Adaptive guidance and integrated navigation with reinforcement meta-learning. *Acta Astronautica*, 169:180–190, apr 2020.

[41] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11):4948, 2021.

[42] Supriyo Ghosh, Sean Laguna, Shiau Hong Lim, Laura Wynter, and Hasan Poonawala. A deep ensemble method for multi-agent reinforcement learning: A case study on air traffic control. *Proceedings of the International Conference on Automated Planning and Scheduling*, 31:468–476, May 2021.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[44] Marc Brittain, Xuxi Yang, and Peng Wei. A deep multi-agent reinforcement learning approach to autonomous separation assurance, 2020.

[45] Songyuan Zhang, Oswin So, Kunal Garg, and Chuchu Fan. Gcbf+: A neural graph control barrier function framework for distributed safe multi-agent control, 2024.

[46] Changxi Zhu, Mehdi Dastani, and Shihan Wang. A survey of multi-agent reinforcement learning with communication. *arXiv preprint arXiv:2203.08975*, 2022.

[47] Kunal Garg, Songyuan Zhang, Oswin So, Charles Dawson, and Chuchu Fan. Learning safe control for multi-robot systems: Methods, verification, and open challenges, 2023.

[48] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. *CoRR*, abs/1605.07736, 2016.

[49] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks, 2018.

[50] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *CoRR*, abs/1805.07733, 2018.

[51] 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.

[52] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *ICLR*, 2020.

[53] Sheng Li, Jayesh K. Gupta, Peter Morales, Ross Allen, and Mykel J. Kochenderfer. Deep implicit coordination graphs for multi-agent reinforcement learning, 2021.

[54] Akshat Agarwal, Sumit Kumar, and Katia P. Sycara. Learning transferable cooperative behavior in multi-agent teams. *CoRR*, abs/1906.01202, 2019.

[55] Cameron Nowzari, Eloy Garcia, and Jorge Cortés. Event-triggered communication and control of networked systems for multi-agent consensus. *Automatica*, 105:1–27, 2019.

[56] Guangzheng Hu, Yuanheng Zhu, Dongbin Zhao, Mengchen Zhao, and Jianye Hao. Event-triggered multi-agent reinforcement learning with communication under limited-bandwidth constraint, 2020.

[57] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Efficient communication in multi-agent reinforcement learning via variance based control, 2019.

[58] Shuai Han, Mehdi Dastani, and Shihan 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.

[59] 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.

[60] 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.

[61] Yuchen Xiao, Weihao Tan, and Christopher Amato. Asynchronous actor-critic for multi-agent reinforcement learning, 2022.

[62] 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.

[63] Rehman Qureshi, Cody Roberts, Emily Kimbrelli, Samuel Mulder, Akhil Rao, Daniel Tauritz, and Davide Guzzetti. A table-top game to simulate competition between p-leo satellite internet constellations. *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 2023.

[64] Fernando Palafox, Yue Yu, and David Fridovich-Keil. Learning hyperplanes for multi-agent collision avoidance in space, 2023.

[65] Daniel Hennes, Daniel Claes, and Karl Tuyls. Evolutionary advantage of reciprocity in collision avodiance. In *AAMAS 2013 Workshop on Autonomous Robots and Multirobot Systems (ARMS 2013)*, 2013.

[66] R. Klima, D. Bloembergen, R. Savani, K. Tuyls, D. Hennes, and D. Izzo. Space Debris Removal: A Game Theoretic Analysis. *Games*, 7(3):20, 2016.

[67] Frank Zagare. A game-theoretic history of the cuban missile crisis. *Economies*, 2(1):20–44, 2014.

[68] Avinash K. Dixit, David McAdams, and Susan Skeath. 'we haven't got but one more day': The cuban missile crisis as a dynamic chicken game.

[69] Karen Pittel and Dirk T. Rübbelke. Transitions in the negotiations on climate change: From prisoner's dilemma to chicken and beyond. *International Environmental Agreements: Politics, Law and Economics*, 12(1):23–39, 2010.

[70] Haimin Hu, Kensuke Nakamura, Kai-Chieh Hsu, Naomi Ehrich Leonard, and Jaime Fernández Fisac. Emergent coordination through game-induced nonlinear opinion dynamics, 2023.

[71] Bryan H Chong. *Extensions to the Chicken Game in the Context of Public Goods*. PhD thesis, 2020.

[72] Marcus Holzinger and Daniel Scheeres. Applied reachability for space situational awareness and safety in spacecraft proximity operations. *AIAA Guidance, Navigation, and Control Conference*, 2009.

[73] Xiaoyu Chen, Gerhard Reinelt, Guangming Dai, and Andreas Spitz. A mixed integer linear programming model for multi-satellite scheduling, 2018.

[74] 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.

[75] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[76] Sydney Dolan, Siddharth Nayak, and Hamsa Balakrishnan. Satellite navigation and coordination with limited information sharing. *Proceedings of Machine Learning Research*, 211:1–14, 2023.

[77] Martin L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[78] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[79] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[80] Juan Luis Gonzalo, Camilla Colombo, and Pierluigi Di Lizia. Analytical framework for space debris collision avoidance maneuver design. *Journal of Guidance, Control, and Dynamics*, 44(3):469–487, March 2021.

[81] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

[82] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[83] 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.

[84] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021.

[85] Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[86] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified massage passing model for semi-supervised classification. *CoRR*, abs/2009.03509, 2020.

[87] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[88] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(56):1633–1685, 2009.

[89] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. pages 544–551, 01 2008.

[90] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[91] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[92] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[93] Johannes Ackermann, Volker Gabler, Takayuki Osa, and Masashi Sugiyama. Reducing overestimation bias in multi-agent domains using double centralized critics. *CoRR*, abs/1910.01465, 2019.

[94] 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.

[95] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. *CoRR*, abs/1803.11485, 2018.

[96] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning. *CoRR*, abs/1706.05296, 2017.

[97] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2019.

[98] V.G. Rao and D.S. Bernstein. Naive control of the double integrator. *IEEE Control Systems Magazine*, 21(5):86–97, 2001.

[99] David A. Vallado and Wayne D. McClain. *Fundamentals of astrodynamics and applications*. Microcosm Press, 2007.

[100] 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.

[101] Ulrich Walter. Orbit perturbations. *Astronautics*, page 555–660, 2018.

[102] Heather Hinkel, Scott P. Cryan, and Chris D'Souza. *Rendezvous and Docking Strategy for Crewed Segment of the Asteroid Redirect Mission*.

[103] Richard Cheng, Mohammad Javad Khojasteh, Aaron D. Ames, and Joel W. Burdick. Safe multi-agent interaction through robust control barrier functions with learned uncertainties. *CoRR*, abs/2004.05273, 2020.

[104] Rose E. Wang, Michael Everett, and Jonathan P. How. R-maddpg for partially observable environments and limited communication, 2020.

[105] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[106] 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.

[107] 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.

[108] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning, 2019.

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

[110] Matteo Gallici, Mario Martin, and Ivan Masmitja. Transfqmix: Transformers for leveraging the graph structure of multi-agent reinforcement learning problems. *arXiv preprint arXiv:2301.05334*, 2023.

[111] 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.

[112] 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.

[113] Xinran Li and Jun Zhang. Context-aware communication for multi-agent reinforcement learning, 2024.

[114] Sydney Dolan, Victor Qin, Geoffrey Ding, and Hamsa Balakrishnan. Satellite collision avoidance using repeated games. *AAS/AIAA Astrodynamics Specialist Conference*, 2023.

[115] Pierluigi Argoneto, Giovanni Perrone, Paolo Renna, Giovanna Lo Nigro, Manfredi Bruccoleri, and Sergio Noto La Diega, editors. *Game Theory: an Overview*, pages 13–24. Springer London, London, 2008.

[116] Hiromu Ito, Yuki Katsumata, Eisuke Hasegawa, and Jin Yoshimura. What is true halving in the payoff matrix of game theory? *PLOS ONE*, 11(8), Aug 2016.

[117] Marie-Laure Cabon-Dhersin and Nathalie Etchart-Vincent. The puzzle of cooperation in a game of chicken: An experimental study. *Theory and Decision*, 72(1):65–87, Jun 2010.

[118] Carmen Pardini and Luciano Anselmo. The short-term effects of the cosmos 1408 fragmentation on neighboring inhabited space stations and large constellations. *Acta Astronautica*, 210:465–473, 2023.

[119] James Lee Foster and Herbert S Estes. *A parametric analysis of orbital debris collision probability and maneuver rate for space vehicles*. NASA Lyndon B. Johnson Space Center, 1992.

[120] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[121] Sean Bruinsma, Thierry Dudok de Wit, Tim Fuller-Rowell, Katherine Garcia-Sage, Piyush Mehta, Fabian Schiemenz, Yuri Y. Shprits, Ruggero Vasile, Jia Yue, and Sean Elvidge. Thermosphere and satellite drag. *Advances in Space Research*, 2023.

[122] Valentin A. Shuvalov, Nikolai B. Gorev, Nikolai A. Tokmak, and Yuri P. Kuchugurnyi. Drag on a spacecraft produced by the interaction of its magnetic field with the earth's ionosphere. physical modelling. *Acta Astronautica*, 166:41–51, 2020.

[123] Giovanni Pecoraro, Ernestina Cianca, Gaetano Marino, and Marina Ruggieri. Preliminary design of a small tracking radar for leo space objects. In *2017 IEEE Aerospace Conference*, pages 1–11. IEEE, 2017.

[124] Janine Liberty. Announcing the mit arclab prize for ai innovation in space. https://aeroastro.mit.edu/news-impact/mit-arclab-prize-for-space-ai-innovation/, Nov 2023.