Reinforcement Learning for Spacecraft Planning and Scheduling

by

Adam Paul Herrmann

B.S., University of Cincinnati, 2019

M.S., University of Colorado, Boulder, 2022

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Aerospace Engineering Sciences

2023

Herrmann, Adam Paul (Ph.D., Aerospace Engineering Sciences)

Reinforcement Learning for Spacecraft Planning and Scheduling

Thesis directed by Prof. Hanspeter Schaub

The coming decades of space exploration will require a massive increase in spacecraft autonomy due to an explosion in the number of Earth-orbiting satellites, which will tax current operations infrastructure and capabilities, and the over subscription of deep space network services for deep space and cislunar missions.

This dissertation investigates the use of reinforcement learning (RL) for spacecraft planning and scheduling, which is the process by which the sequence of tasks a spacecraft must execute to achieve its objectives is computed. RL is a machine learning technique that allows for autonomous decision-making agents to learn an optimal policy that maps situations to actions to maximize a numerical reward function. RL offers closed-loop decision-making, fast execution times after training, and few constraints on problem representation.

This dissertation first investigates the application of RL to the single satellite Earth-observing scheduling problem, taking into consideration various spacecraft resources, data downlink, and agile targeting of surface targets. RL-based formulations and methods are shown to meet or exceed the performance of genetic algorithms and generalize over the state space. Then, scalable Earth-observing constellation operations utilizing single-agent RL policies are investigated. Various communication assumptions and target distribution methods are explored. A novel fully decentralized RL-based architecture that can automatically adjust to a new constellation design or new distribution of surface targets is developed and shown to be more performant than a centralized architecture that relies on an integer program for target distribution. Finally, RL is applied to the problem of small body science operations, demonstrating that RL is capable of autonomously managing maneuvers, navigation updates, resources, and science operations to accomplish a mission.

Dedication

Robert V. Stevens.

Acknowledgements

First and foremost, I'd like to thank Dr. Schaub for his dedication, support, and availability as an advisor. Thank you for giving me a shot as a PhD student. My time in the AVS Laboratory has changed the trajectory of my life, and I can't wait to incorporate all of the things I have learned from you into my future career.

I would like to thank several members of the AVS Laboratory as well. Thank you to Andrew Harris, who began the AI work in the AVS Laboratory. He's a large reason as to why I've had an excellent research topic. I'd also like to thank John Martin. They say that you should always surround yourself with people smarter than you. John is the embodiement of this statement, and I am constantly using his suggestions in my workflow and research. Finally, I'd like to thank Mark Stephenson who quickly became a valuable colleague in the AVS Laboratory.

I would also like to thank my partner Dana Chafetz for her unwaivering support and patience during the PhD process. Thank you to our cat, Tuna, for providing entertainment and companionship during the process. Thank you to my parents, Paul and Jennifer Herrmann, for your support and encouragement throughout the whole process as well.

Finally, I would like to thank the organizations that have funded my PhD. Thank you to the NASA Space Technology Graduate Research Opportunities (NSTGRO) program that has funded my PhD for the past three years, and thank you to Ben Hockman for being an excellent NASA mentor. I'd also like to thank the Air Force Research Laboratory (AFRL) for their support via the Small Business Technology Transfer (STTR) program. Finally, thank you to CU Research Computing (CURC) for the computational resources that have enabled much of this work.

Contents

Tables

Figures

Figure

Chapter 1

Introduction

1.1 Motivation

As the cost of spaceflight decreases thanks to low-cost launch providers, the miniaturization of space technology, and renewed interest in the commercialization of space, the number of spacecraft in orbit is increasing exponentially. As of May 2022, approximately 5,000 operational satellites exist in orbit [2]. A survey in 2019 concluded that over 20,000 satellites were announced to be launched into orbit over the next decade [3]. SpaceX alone plans to operate at minimum a constellation of 12,000 Starlink satellites in low Earth orbit to provide global low latency broadband internet [4]. Other companies that aim to provide global internet coverage include OneWeb, Telesat, and Amazon. Each of these constellations will be composed of thousands of spacecraft [5]. The global internet market is not the only one that has seen renewed interest. Many companies are now operating constellations of Earth-observing satellites, which provide a variety of imaging products to government, academic, and industry partners. Planet, Spire Global, and Capella Space are just a few examples of such companies. An example of one such constellation is provided in Figure 1.1. This exponential increase in the number of satellites in orbit will necessitate increased capabilities for spacecraft autonomy, from autonomous maneuver management to autonomous science collection and resource management on board spacecraft.

Earth is not the only domain in which spacecraft will benefit from increased autonomous capabilities. Both cislunar and deep space will require increased autonomy for a multitude of reasons. The round-trip light-time delay is the amount of time it takes for a signal to leave the Earth,

Figure 1.1: Spire Global Low Earth Multi-Use Receiver (LEMUR) CubeSat constellation. Credit: Spire Global.

reach a spacecraft, and return to Earth. In deep space, the round-trip light-time delay is tens of minutes depending on the distance from the Earth to the spacecraft in question. Spacecraft cannot rely on input from Earth for unexpected events that occur during this time. To make matters worse, a NASA Inspector General report found that the Deep Space Network (DSN), which responsible for communication and navigation for 60 NASA and international space missions, is currently oversubscribed and will remain oversubscribed as NASA's Artemis Program, Perseverance Rover, and James Webb Space Telescope compete for DSN access [6]. Future missions will need to integrate advanced autonomous capabilities to reduce their reliance on the DSN. Finally, missions in deep space are faced with uncertain and unexplored environments. As long as humanity is constantly pushing the boundaries of exploration, there will be an unavoidable amount of epistemic uncertainty that must be addressed during operations. Spacecraft must be able to reduce and respond to this uncertainty without relying on Earth for input. When each of these challenges are considered, it is clear that increased autonomous capabilities are a must for future deep space missions.

In the context of spacecraft operations, planning and scheduling is the process by which the sequence of commands a spacecraft must execute to fulfill its mission objectives is computed. Examples of planning and scheduling problems include:

(1) An Earth-observing satellite (EOS) scheduling a sequence of observation and downlink tasks to maximize the amount of science data collected

(2) A spacecraft planning a sequence of maneuvers and science operations about an asteroid to map the body with various instruments and determine a landing site

Autonomous, on-board planning and scheduling capabilities can help address some of the challenges facing the space industry in the coming decades. First, autonomous planning and scheduling can reduce the burden on spacecraft operators. Currently, spacecraft operators must generate a plan on the ground and uplink the plan to the spacecraft. This process can take anywhere from several minutes to several hours, depending on the complexity of the problem, the selected solution method, and the round-trip light-time delay. If the spacecraft is able to generate a plan on board, the burden on spacecraft operators and operations infrastructure is reduced. Second, autonomous planning and scheduling can enable opportunistic science. Following a ground-based planning paradigm, if a spacecraft detects an interesting science event, it is unable to react to the event and must wait for the next planning cycle. If the spacecraft is able to generate a plan on board the spacecraft, it can react to the event and collect additional science data and even coordinate on the fly with other spacecraft in the vicinity. Finally, autonomous planning and scheduling can increase the robustness of a plan. If the spacecraft is able to generate a plan on board the spacecraft, it can react to unexpected changes in the environment, such as a ground station outage or a thruster failure.

However, on-board planning and scheduling for spacecraft is a challenging endeavor for several reasons. The environment is dynamic, meaning that the state of the environment and the spacecraft evolves with time, due in part to the actions of the spacecraft. In the Earth-observing satellite example, the ground targets available to the spacecraft are constantly changing, and the resources on board the spacecraft are impacted by the actions of the spacecraft. Autonomous decision-making agents must be able to take into account the dynamic nature of the environment. The environment is also uncertain. The spacecraft must observe the state of the environment with noisy sensors, and not all states are observable. For deep space missions, this can be especially challenging. A belief state of the environment must be constructed using these measurements and our knowledge of how the state of the environment and spacecraft evolves with time. The decision-making agents must be able to account for this uncertainty in the planning problem and address it with whatever means are available. Finally, the state and decision space are highly dimensional. Determining which information is necessary to make decisions autonomously is non-trivial, and the number of possible decisions is large. It is desirable to keep the state and decision space as small as possible to reduce the computational complexity of the planning and scheduling problem. However, the state and decision-space must be large enough to capture the relevant features and complexity of the problem, allowing solution methods to produce performant plans.

To address the challenges facing the future of space exploration, and to enable autonomous, on-board planning and scheduling capabilities, this dissertation investigates the use of reinforcement learning (RL) for spacecraft planning and scheduling. Reinforcement learning is a class of problem formulations and algorithms that generate autonomous decision-making agents through repeated interaction with a real or simulated environment with the goal of maximizing a reward signal. RL is an excellent choice for on-board planning and scheduling due to minimal restrictions on the problem representation and cost function, the ability to learn optimal behavior from experience, and the fast execution times of trained agents. In recent years, reinforcement learning has been shown to achieve superhuman performance in several challenging problems including the game of Go [1], Atari games [7], and Dota 2 [8]. This dissertation aims to help bring about the ubiquitous adoption of reinforcement learning for spacecraft planning and scheduling by investigating how reinforcement learning can be applied to planning and scheduling problems in Earth-orbiting and deep space domains.

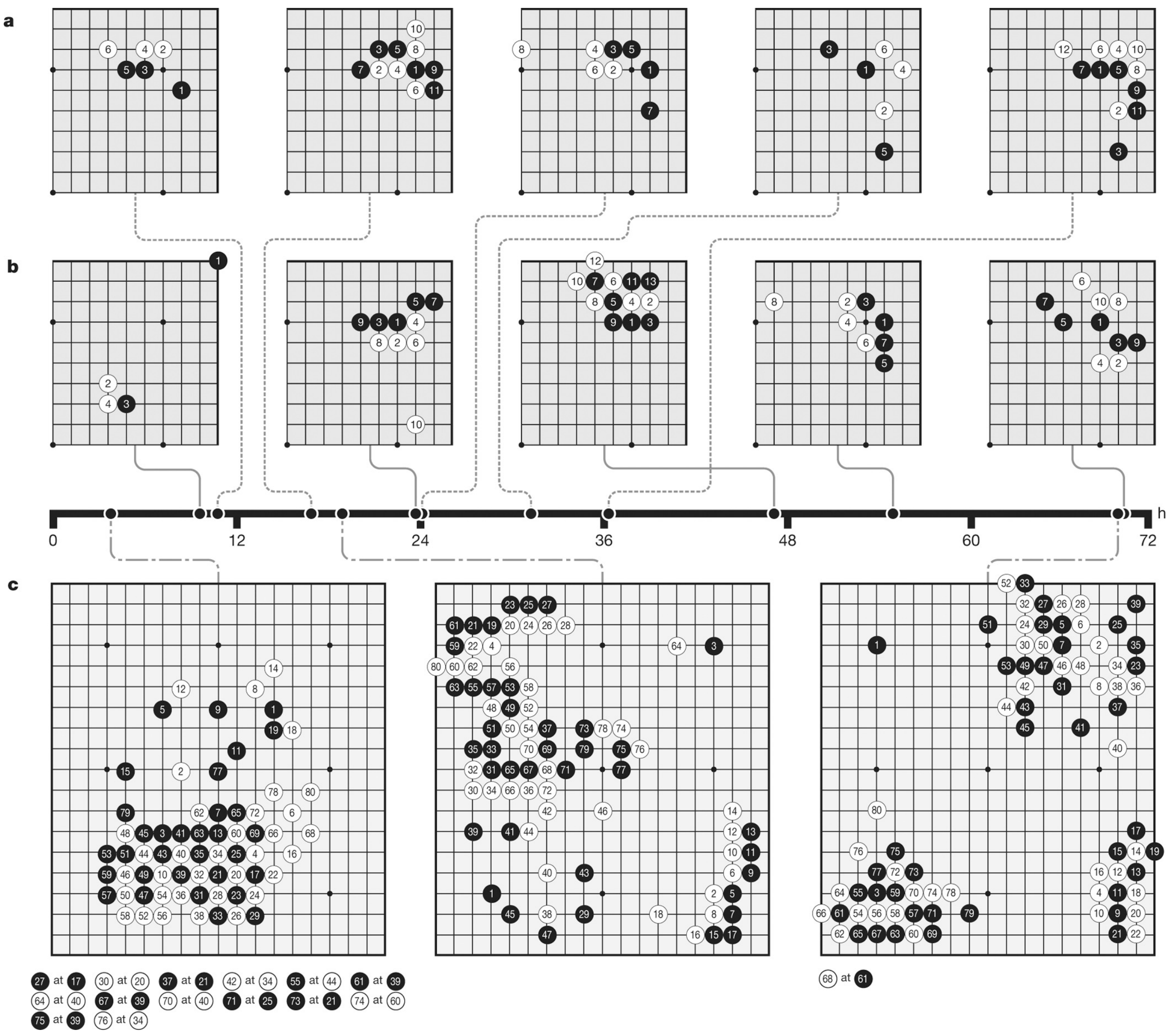


Figure 1.2: Evolution of Go sequences learned by AlphaGo Zero. Credit: Silver et al. [1].

1.2 Related Work

1.2.1 Traditional Planning and Scheduling

Spacecraft planning and scheduling is traditionally a ground-based process. During the early phases of mission development, and on a periodic basis during operations, an iterative process between science planning and mission planning occurs to define the science objectives and spacecraft trajectory at different levels of fidelity. These two items are then input into an activity planner,

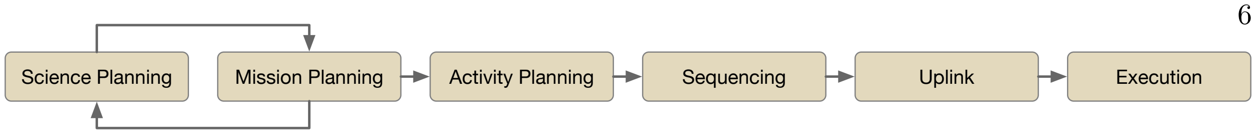


Figure 1.3: Traditional planning and scheduling process.

which generates an activity plan detailing the tasks the spacecraft must complete to meet the science objectives. The activity plan may also take into account inputs from navigation and flight dynamics teams, depending on the mission in question. After the activity plan is generated, it is sequenced into commands, which are then uplinked to the spacecraft and executed open-loop on board the spacecraft. This process is depicted in Figure 1.3.

This approach to spacecraft planning and scheduling introduces several challenges. First and foremost, the plan is executed open-loop. This means that the spacecraft does not have the ability to react to changes in the environment. If the spacecraft is unable to complete a task, or if the spacecraft is unable to meet a constraint, the plan must be re-generated on the ground and uplinked to the spacecraft. Again, this process can take anywhere from several minutes to several hours. Furthermore, this approach is not extensible to opportunistic science events. If the spacecraft detects an interesting science event, it is unable to react to the event and must wait for the next planning cycle to react. Finally, this approach is not robust to changes in the environment, which may require that the plan is re-generated on the ground and uplinked to the spacecraft. This is not a problem for missions with a high level of predictability, such as Earth-observing satellites. However, for missions with a relatively large amount of uncertainty, such as deep space missions, robustness must be included in the spacecraft plan, which may come at the cost of science return, efficiency, or science quality. Therefore, autonomous on-board planning and scheduling algorithms are desirable to reduce the time required to react to changes in the environment, to enable opportunistic science, and to increase the robustness of the plan.

Significant effort has been made to address these challenges by giving the spacecraft the ability to modify the ground-based plan in the event of a contingency. CASPER is an on-board planning and scheduling software developed by the National Aeronautics and Space Administration

(NASA) that utilizes a technique known as iterative repair, which continually checks an existing plan for resource constraint violations and modifies the plan on-board the spacecraft if necessary [9]. CASPER, in addition to an automated ground-based planning tool named ASPEN [10], have been applied to several missions, such as EO-1 [11, 12] and IPEX [13], to demonstrate on-board schedule modification of ground-based plans. These systems operate in combination with science detection algorithms and sensor webs to mark future science targets [14, 15]. MEXEC is another on-board planning and execution tool originally developed by NASA for the Europa Clipper mission [16]. An on-board scheduler is currently being flown on the Perseverance Rover to adjust activities to account for variation in resources or task execution [17]. The Scheduling Planning Routing Inter-satellite Network Tool (SPRINT) developed by the Massachusetts Institute of Technology (MIT) takes a similar approach to address the replanning problem by using a global planner for scheduling a constellation of Earth-observing satellites and on-board planners on each satellite for unexpected opportunities [18, 19].

While on-board replanning tools address many of the challenges associated with ground-based planning and scheduling, they require an apriori plan as an input. This plan is still generated on the ground and uplinked to the spacecraft on a periodic basis, which is a time-consuming process during regular operations. Providing spacecraft with even more control over their operational decisions can reduce the burden on spacecraft operators, saving time and money.

1.2.2 Optimization-Based Planning and Scheduling

In addition to the challenges introduced by a ground-based planning and scheduling paradigm, the solution method selected for the activity planning block can introduce even more challenges. Optimization-based approaches to scheduling utilize heuristic, metaheuristic, or exact solution methods [20]. Metaheuristic optimization algorithms such as genetic algorithms (GAs) have successfully been applied to Earth-observing scheduling problems [21]. Genetic algorithms do not place constraints on the fitness function. For instance, the fitness function could be a high-fidelity, nonlinear spacecraft simulation that returns a cost function reflecting the number of collected and downlinked targets over the planning horizon. However, metaheuristic algorithms are sensitive to initialization, often times do not guarantee global optimality, and can be slow to converge (especially if a high-fidelity simulator is used). Mixed integer linear programming approaches are perhaps the most popular in the literature due to optimality guarantees and speed of convergence after the relevant data is pre-processed [22, 23, 24, 25, 26, 27]. Spire Global [28] and Planet [29] utilize mixed integer linear programming for their Earth-observing constellations. However, the linearity requirements of MILP formulations prohibits the accurate representation of nonlinear planning problems. Some resources, such as wheel speeds, are not well modeled with a linear approach. While the dynamics can be linearized, this requires a small enough discretization of the planning horizon, which causes the number of decision variables to explode. Mixed integer nonlinear programming (MINLP) formulations can certainly address these issues head-on, but many MINLP formulations are difficult to solve, especially if the problem is non-convex [30]. Furthermore, even if these issues are addressed, the computational cost of pre-processing the data can be significant. For instance, access times between spacecraft and ground stations must be pre-computed. If a large planning horizon or large number of ground stations is used, the pre-processing time can be too great to run these algorithms on-board spacecraft.

1.2.3 Reinforcement Learning Approaches

Reinforcement learning (RL) has recently emerged as a potential solution method for spacecraft planning and scheduling. In comparison to the aforementioned optimization methods, reinforcement learning has the ability to leverage domain randomization to deal with model uncertainty [31]. Furthermore, reinforcement learning places limited constraints on the model used to represent the problem, which is a major benefit over exact solution methods like mixed integer programming. The objective of reinforcement learning is to solve for a policy that maps states to actions to maximize a numerical reward function [32]. Once the optimal policy has been solved for, the action that maximizes the reward function for any state is known. For simple RL problems with a discrete state and action space or a discretized continuous state and action space, this policy can be represented

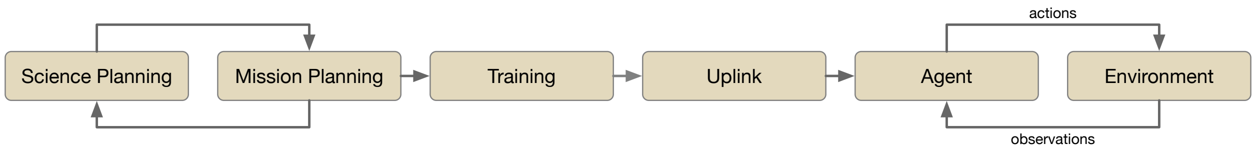


Figure 1.4: Reinforcement learning-based planning and scheduling.

in the form of a lookup table. However, for complex planning problems with a continuous state and action space, the policy must be represented in the form of a function approximator. Function approximators can take many forms, such as neural networks, decision trees, or Gaussian processes. Neural networks are the most popular form of function approximator for reinforcement learning. Once trained, neural networks can be executed in near real-time, making them a promising solution method for on-board planning and scheduling. References [33] and [34] demonstrate the execution of neural networks on flight processors (or processors currently undergoing validation for flight), ranging from milliseconds to seconds of execution time. Due to fast execution times, a low memory footprint, the potential for optimal policies with respect to the reward function, and flexibility in the choice of problem representation, reinforcement learning is an excellent candidate for both low- and high-level spacecraft autonomy. A diagram of reinforcement learning-based planning and scheduling is provided in Figure 1.4. The "activity planning" block is replaced with a "training" block, and decision-making agent now executes the policy learned by the reinforcement learning algorithm in closed-loop fashion on-board the spacecraft.

Much of the literature investigating reinforcement learning for spacecraft decision-making problems falls within the domain of the guidance, navigation, and control (GNC) subsystem. Reinforcement learning has been used for planetary landing [35, 36, 37, 38], small body proximity operations [39, 40, 41, 42], and spacecraft rendezvous, proximity operations, and docking (RPOD) [43, 44, 45, 46, 47, 48]. Many of these works focus on using reinforcement learning algorithms that can adapt to off-nominal conditions on the fly, such as thruster failures, and still successfully complete the mission and/or use reinforcement learning to provide end-to-end solutions for some or all aspects of the guidance, navigation, and control subsystem. A popular approach is the use of recurrent policies trained with reinforcement learning algorithms that maintain an internal belief state to handle the partial observability of the associated problem. References [36], [38], [39], [40], [42] integrate the full guidance, navigation, and control subsystem into a reinforcement learning-only framework (i.e. they map observations to controls) using this approach. References [35], [45], [47], [46], and [37] assume full observability over the state, while Reference [41] assumes that a state estimate from a navigation subsystem is provided. While there are many interesting GNC problems reinforcement learning can solve, this work focuses on planning and scheduling. The aforementioned works may incorporate some aspects of planning and scheduling (e.g. maneuver sequencing, maneuvering for science purposes, etc.), but are not planning and scheduling problems due to the lack of focus on science objectives and the lack of resource constraints and management. This work treats the GNC subsystem as an input into the problem formulation as opposed to being the problem formulation itself.

In addition to solving GNC problems, reinforcement learning has been applied to many Earth-observing satellite (EOS) scheduling problems. However, each paper solves a different problem using a different algorithm, such as Asynchronous Advantage Actor-Critic (A3C) [49], REINFORCE [50], and Deep Q-Networks (DQN) [51]. Furthermore, these works often insufficiently model resources such as battery charge, data storage availability, and reaction wheel speeds, as well as their impact on the EOS scheduling problem. Harris et al. demonstrate utilizing Shielded Proximal Policy Optimization (SPPO) for EOS scheduling with battery and reaction wheel speed constraints. SPPO bounds the decision-making agent's actions during training and deployment such that only safe actions are taken [52]. Shielded deep reinforcement learning utilizes a linear temporal logic specification to monitor the actions output by the policy, overriding the actions if they violate the LTL specification [53]. This approach is also taken by Dunlap et al. in Reference [48] for safe spacecraft docking. Eddy and Kochenderfer apply Monte Carlo tree search (MCTS) to a semi-Markov Decision Process (SMDP) formulation of the EOS scheduling problem with battery and data storage resource constraints, demonstrating near-optimal performance compared to a mixed integer programming formulation and solution [54]. Monte Carlo tree search is an online reinforcement learning algorithm that searches over the state and action space to determine the best action to take at each decision-making interval [55]. While there has been much work on EOS scheduling with reinforcement learning, there is no standard of comparison for different problem formulations and algorithms. Each paper solves its own flavor of EOS scheduling problem, and typically do not compare RL methods to one another.

1.3 Summary of Objectives

In contrast to prior work, this dissertation systematically explores various EOS scheduling problems with far more realistic problem dynamics and resource constraints. In addition to the power and reaction wheel speed resources investigated by Harris and Schaub [52], this dissertation investigates the addition of on-board data storage and data downlink, which can have serious implications on learned policies. Storage limitations and downlink availability restricts when agents can and cannot perform science activities. This thesis also investigates more challenging science objectives, such as agile targeting of imaging targets. In the agile EOS scheduling problem, a spacecraft with three-axis attitude control capabilities targets various imaging targets in different along- and cross-track directions. The inclusion of hundreds or thousands of targets that may be individually imaged impacts the size of the decision space, causing an explosion in the problem complexity and requiring that care is taken when formulating the problem and deploying a solution method such that training decision-making agents is tractable. This dissertation also addresses the lack of standardization in the reinforcement learning literature for EOS scheduling by providing common problem formulations and comparing the performance of different reinforcement learning algorithms for these formulations. A comprehensive comparison between deep reinforcement learning, Monte Carlo tree search, and genetic algorithms is performed to determine which algorithm is best suited for different classes of EOS scheduling problems.

This thesis then explores multi-satellite agile Earth-observing constellation operations, which also can result in an explosion in problem complexity because the decision space is exponential in the number of decision-making agents. Once again, care must be taken to ensure the problem is tractable to solve. Furthermore, solution methods must be scalable, allowing for the addition and removal of satellites in the constellation without requiring retraining. To address these challenges, the decision-making agents trained for the agile EOS scheduling problem are deployed in various Walker-delta constellations with various cross-link communication assumptions. Higher-level coordination among the decision-making agents is investigated by using a centralized target distribution method, which utilizes mixed integer programming to distribute the imaging targets among the constellation of spacecraft.

Finally, this thesis investigates reinforcement learning for small body science operations problems, which are largely unexplored in the literature. Instead of studying the application of RL to the GNC subsystem as past work does, this work treats the GNC subsystem as an input into the problem formulation. The decision-making agent sets the attitude reference, translational reference state about the small body, and the behavior of the navigation system. In addition to the problem features of EOS planning and scheduling, small body science operations includes translational guidance and control due to the weak gravity around small bodies. A relatively large amount of state uncertainty is also present in the problem, necessitating the handling of this uncertainty in the problem formulation and solution method. The tools developed for and the lessons learned from EOS scheduling are applied to small body science operations in order to develop robust and performant decision-making agents that can autonomously operate on board spacecraft in the presence of state uncertainty.

The objectives of this dissertation may be summarized as follows:

(1) **Single Satellite Earth-Observing Scheduling:** Formulate and solve Earth-observing satellite scheduling problems as Markov decision processes with various science objectives and resource constraints using deep reinforcement learning, Monte Carlo tree search, and genetic algorithms. This work may be found in Chapters 3, 4, and 5.

(2) **Multi-Satellite Earth-Observing Scheduling:** Investigate the use of single agent reinforcement learning to enable scalable EOS constellation operations considering various

cross-link communication assumptions and target distribution methodologies. This work may be found in Chapter 6.

(3) **Small Body Science Operations:** Leverage the lessons learned in EOS planning and scheduling to formulate and solve small body science operations problems as Markov decision processes, considering translational guidance and control and state uncertainty in the problem formulation. This work may be found in Chapters 7 and 8.

Chapter 2

Reinforcement Learning

This chapter provides an overview of some fundamental concepts in reinforcement learning. An overview of Markov decision processes, how to solve Markov decision processes, and reinforcement learning algorithms is provided. The material in this chapter summarizes the core concepts found in References [32], [55], and [56], which can provide more detailed explanations for those interested.

2.1 Markov Decision Processes

Reinforcement learning problems are formulated as Markov decision processes (MDPs), sequential decision-making problems in which an agent observes some state s_i and selects and action a_i following a policy $\pi : \mathcal{S} \to \mathcal{A}$, which maps states to actions. The set of all possible states is referred to as the state space $\mathcal{S}$, and the set of all possible actions is referred to as the action space $\mathcal{A}$. The agent observes a new state s_{i+1} and receives a reward r_i based on the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$. This process is demonstrated in Figure 2.1. Markov decision processes follow the Markov assumption, meaning the next state is conditionally dependent only on the current state and action. Mathematically, this may be stated as $T(s_{i+1}|s_i, a_i) = T(s_{i+1}|s_i, a_i, s_{i-1}, a_{i-1}, ..., s_0, a_0)$. The transition function can also be represented with a generative model, which integrates equations of motion or samples an underlying distribution to return a new state: $s_{i+1} \sim G(s_i, a_i)$.

The return is referred to as the sum of all rewards. For a finite-horizon problem with N

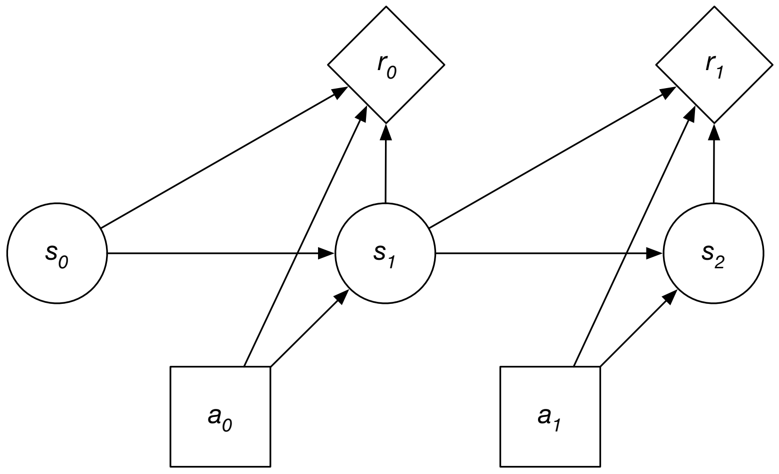


Figure 2.1: Markov decision process. A decision-making agent takes an action a_i while in state s_i. The decision-making agent transitions to a new state s_{i+1} and receives a reward r_i.

decision-making intervals, the return is given as:

$$\sum_{i=0}^{N} r_i \tag{2.1}$$

In many cases, an infinite-horizon problem formulation is more appropriate. In this case, however, the previous definition of the return can result in infinite returns. To avoid this, the return is discounted by a discount factor, $\gamma \in [0, 1)$. The discounted return is given as:

$$\sum_{i=0}^{\infty} \gamma^i r_i \tag{2.2}$$

The goal of reinforcement learning is to find a policy π that maximizes the expected return. The expected return is also referred to as the value function $V(s)$. The value function associated with some policy π when starting in some state s_i is given in Equation 2.3, where $\mathbb{E}$ is the expected value operator. The value function can be thought of as a measure of how "good" a particular state is.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{i+k} \,\middle|\, s_i = s\right] \tag{2.3}$$

A state-action value function may also be defined, which is the expected return when starting in some state s_i, taking some action a_i, and following some policy π thereafter. This is given in Equation 2.4. This is useful when evaluating the value associated with a particular state-action pair. Similar to the value function, the state-action value function can be thought of as a measure of how "good" a particular state-action pair is.

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{i+k} \,\middle|\, s_i = s, a_i = a\right] \tag{2.4}$$

2.2 Tabular Solution Methods

Tabular solution methods solve for the optimal value function or policy in tabular form, which means that the value function or policy is represented with an array, a dictionary, or lookup table. These methods are most useful for small problems, particularly those with discrete state and action spaces. Furthermore, these methods introduce fundamental concepts used in more advanced solution methods.

2.2.1 Optimal Policies and Value Functions

Solving Markov decision processes involves solving for the optimal policy π^*, which maximizes the expected return. The optimal policy is given in Equation 2.5.

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s) \tag{2.5}$$

The optimal value function is the value function associated with following the optimal policy, and is referred to as $V^*(s)$. The optimal value function can be defined recursively using the Bellman optimality equation, given in Equations 2.6 and 2.7.

$$V^*(s) = \max_{a} \sum_{s_{i+1}\in\mathcal{S}} T(s_{i+1}|s_i,a)\left[R(s_i,a) + \gamma V^*(s_{i+1})\right] \tag{2.6}$$

$$Q^*(s,a) = \sum_{s_{i+1}\in\mathcal{S}} T(s_{i+1}|s_i,a)\left[R(s_i,a) + \gamma \max_{a_{i+1}} Q^*(s_{i+1},a_{i+1})\right] \tag{2.7}$$

If the optimal value function has been solved for, the optimal policy can be extracted from the optimal value function by using the expression in Equation 2.8.

$$\pi^*(s) = \arg\max_{a} \sum_{s_{i+1}\in\mathcal{S}} T(s_{i+1}|s_i,a)\left[R(s_i,a) + \gamma V^*(s_{i+1})\right] \tag{2.8}$$

If the optimal state-action value function is known, this can be reduced to:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{2.9}$$

2.2.2 Dynamic Programming Methods

The question of exactly how the optimal value function or state-action value function can be solved for still remains. If the transition probabilities and reward function are known, and if the state and action spaces are discrete, the optimal value function can be solved for using a variety of tabular solution methods that fall under the umbrella of dynamic programming. One such technique is called value iteration. Value iteration is an iterative algorithm that begins with an initial guess of the value function, which is continually updated by enumerating through the state space and applying the Bellman optimality equations. Value iteration is guaranteed to converge to the optimal value function given enough iterations regardless of the initial guess. The value iteration algorithm is provided in Algorithm 1.

Algorithm 1 Value Iteration

```
1: $k = 0$
2: Initialize $V^k(s)$ arbitrarily for all $s \in \mathcal{S}$
3: **repeat**
4:     **for** $s \in \mathcal{S}$
5:         $V^{k+1}(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a) \left[R(s,a) + \gamma V^k(s')\right]$
6:         $k \leftarrow k+1$
7: **until** convergence
```

Similar to value iteration, policy iteration is an algorithm that can solve for the optimal policy directly, provided that the same assumptions are made. However, the discrete state and action space assumption can be difficult to apply to real world problems. Continuous state and action spaces could be discretized, but this may not be feasible for large problems. Furthermore, the dynamics of many problems cannot be easily modeled with an explicit transition function. While there are techniques that can leverage linear dynamics and quadratic reward functions, these assumptions also may not apply to real world problems.

2.2.3 Online Methods

Online methods are a class of tabular methods that interact with a model of the environment in order to learn an optimal policy or value function. Unlike the methods in the previous section, online methods restrict computation to states that are reachable from the current state in the environment, which is beneficial for problems with large state and action spaces. Examples of online methods include forward search and branch and bound. Both methods construct a search tree over the state and action spaces to search for the optimal policy. However, both of these methods require explicit transition functions to compute the value of each node in the search tree.

To avoid these issues, one can use sampling-based online methods that only require a generative model of the transition function, $s_{i+1} \sim G(s_i, a_i)$. One such method is called Monte Carlo Tree Search (MCTS). MCTS is an online method that uses a generative model of the transition function to incrementally step through the environment by constructing a search tree over the state and action spaces. At each step through the environment, MCTS runs a number of iterations to refine $\hat{Q}(s, a)$, an estimate of the state-action value function. At each iteration, the algorithm selects an action to transition to a leaf node in the search tree. The action is selected based on the current estimate of the state-action value function and an exploration term. The exploration term, $U(s, a)$, is computed using the number of times the state has been visited, $N(s)$, and the number of times the action has been taken from the state, $N(s, a)$, as shown in Equation 2.10.

$$U(s,a) = \epsilon\sqrt{\frac{\log(N(s)+1)}{N(s,a)+1}} \tag{2.10}$$

This step is referred to as the selection step. If the state has not been visited before, the algorithm then expands the leaf node by adding a child node for each possible action. The estimate of the state-action value function is initialized to some initial value when this happens. This is referred to as the expansion step. In the next step, the rollout step, the algorithm then executes a rollout policy until the specified depth, d, is met or the episode terminates. The return of this trajectory is then used to update the value of the child node that the rollout execution began from. MCTS then backpropagates the value of the child node all the way to the parent node in the backup step.

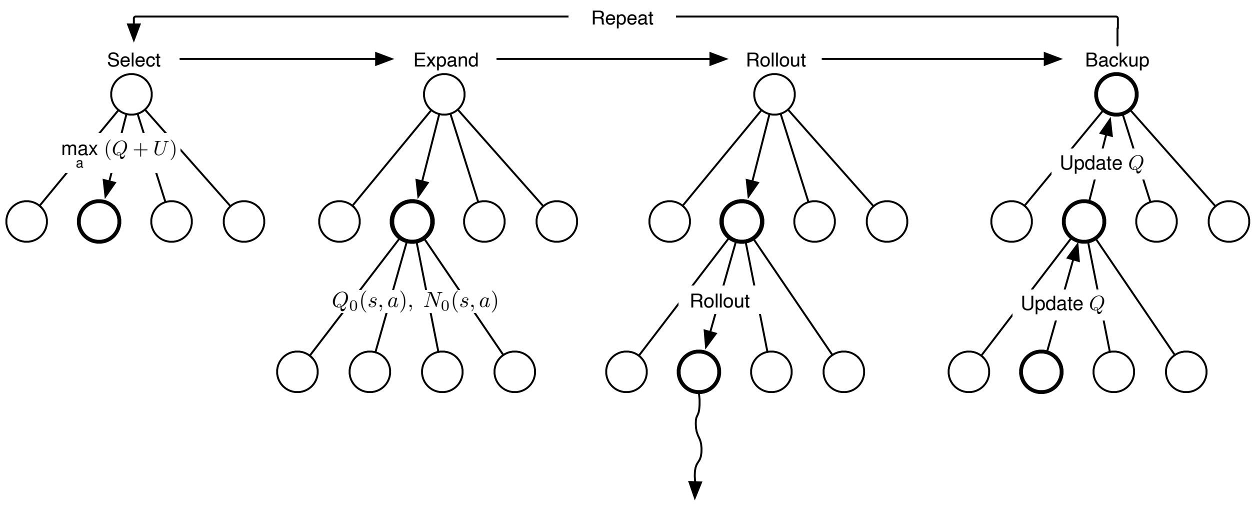


Figure 2.2: Monte Carlo tree search algorithm. In the **selection** step, MCTS selects the action that maximizes the state-action value estimate and an exploration term. In the **expansion** step, MCTS initializes the state-action value function and the number of times each state-action pair has been selected. During **rollout**, MCTS uses a rollout policy to select actions until a specified depth. Finally, during **backup**, MCTS updates the state-action value estimates with the return from rollout.

A diagram of this process is provided in Figure 2.2.

The full algorithm for MCTS is provided in Algorithm 2. In the main routine, the set of visited states T is initialized to an empty set. The initial state is also set. Then, for some specified number of steps K, the SIMULATE function is called, which iterates N times to refine the estimate of the state-action value function before returning the best action. The environment is then stepped through using this action, and the process repeats.

Monte Carlo tree search is guaranteed to converge to the optimal action as the number of simulations-per-step approaches infinity [57, 58]. MCTS is also at the center of the AlphaGo Zero algorithm, which achieved superhuman performance in the game of Go [1]. However, AlphaGo Zero implements a number of improvements to the basic MCTS algorithm, including the use of a neural network to guide the search process and generalize over the state space. One of the shortcomings of MCTS is its inability to generalize over the state space as it is only concerned with building a search tree based on reachable states. In the event that a different initial condition is used, the entire algorithm must be run again, which can be expensive.

Algorithm 2 Monte Carlo Tree Search

```
function SELECT_ACTION($s$, $d$, $N$)
    for $1 : N$
        SIMULATE($s$, $d$, $\pi$)
    return $\arg\max_a \hat{Q}(s, a)$

function SIMULATE($s$, $d$, $\pi$)
    if $d = 0$
        return 0
    if $s \notin T$
        for $a \in \mathcal{A}$
            $N(s, a) \leftarrow 0$
            $\hat{Q}(s, a) \leftarrow 0$
        $T \leftarrow T \cup \{s\}$
        return ROLLOUT($s$, $d$, $\pi$)
    $a \leftarrow \arg\max_a \hat{Q}(s, a) + \epsilon\sqrt{\frac{\log(N(s)+1)}{N(s,a)+1}}$
    $q \leftarrow r + \gamma$SIMULATE($G(s, a)$, $d - 1$, $\pi$)
    $N(s, a) \leftarrow N(s, a) + 1$
    $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \frac{q - \hat{Q}(s, a)}{N(s, a)}$
    return $q$

function ROLLOUT($s$, $d$, $\pi$)
    if $d = 0$
        return 0
    $a \sim \pi(s)$
    $s', r \sim G(s, a)$
    return $r + \gamma$ROLLOUT($s'$, $d - 1$, $\pi$)

main routine:
    $T \leftarrow \emptyset$
    $s \leftarrow s_0$
    for $i = 1 : K$
        $a \leftarrow$ SELECT_ACTION($s$, $d$, $N$)
        $s', r \sim G(s, a)$
        $s \leftarrow s'$
```

2.3 Deep Reinforcement Learning

2.3.1 Function Approximation

A benefit to the dynamic programming algorithms discussed is that they solve for the value function or policy over the entire state and action space. However, these algorithms require a discrete state and action space as well as an explicit transition function. These types of algorithms are referred to as tabular methods by Sutton and Barto and cover Monte Carlo methods (not to be confused with Monte Carlo tree search), temporal-difference methods, and dynamic programming [32]. MCTS is also considered to be a tabular method, but only solves for the value function or policy over the reachable state space using a generative transition function. Only solving over reachable states has its upsides in terms of computation, but also comes with a significant drawback, as it means that the algorithm must be run again if the initial state is changed.

Approximate solution methods attempt to remedy these issues with the use of universal function approximators. These methods are able to generalize over the state space by learning from only a subset of the state space. Universal function approximators parameterize the value function or policy using a set of parameters θ. In this dissertation, the parameterized value functions is referred to as $V_\theta(s)$ and the parameterized policies are referred to as $\pi_\theta(s)$. Many types of function approximators exist, but the most popular for reinforcement learning is the artificial neural network (ANN), a nonlinear function approximator made up of a series of interconnected layers that are meant to model the neurons in a nervous system. A diagram of an artificial neural network is provided in Figure 2.3. Each node in the ANN represents an activation function, and the lines represent the edges, or outputs, from one node to another. Each edge has its own weight and bias that determines the strength of the signal from the corresponding node.

An artificial neural network can be defined as the function g. The input vector is $\mathbf{x}$, the target output vector is $\mathbf{y}$, and the predicted output vector is $\hat{\mathbf{y}}$. The activation function is also referred to as f. Assuming we have L hidden layers, the predicted outputs computed by the network are:

$$\hat{\mathbf{y}} = g(\mathbf{x}) = f_L(\mathbf{W}_L f_{L-1}(\mathbf{W}_{L-1} \cdots f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_{N-1}) + \mathbf{b}_K) \tag{2.11}$$

where $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weights and biases for the lth layer (collectively referred to as θ). In reinforcement learning, the input vector $\mathbf{x}$ is the state of the MDP, s. The target output vector $\mathbf{y}$ is the value function $V(s)$, the state-action value function $Q(s, a)$, or the policy $\pi(s)$. The predicted output vector $\hat{\mathbf{y}}$ is the parameterized value function $V_\theta(s)$, the parameterized state-action value function $Q_\theta(s, a)$, or the parameterized policy $\pi_\theta(s)$.

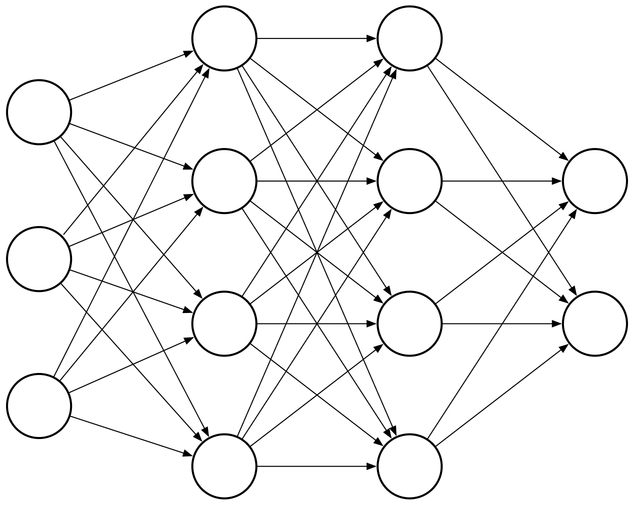


Figure 2.3: Diagram of an artificial neural network. This ANN has three inputs, two hidden layers of four nodes each, and two outputs.

During training, the weights and biases of the neural network are updated incrementally over many iterations. During each iteration, a forward pass of the network is performed, where the predicted output $\hat{\mathbf{y}}$ is computed. Then, a loss computation is performed using the loss function, J, which represents the error between the predicted output and the target output. Popular loss functions include mean squared error or mean absolute error. Next, the backpropagation step is performed, where the gradients of the loss function with respect to the weights and biases are computed by recursively applying the chain rule. Finally, the weights and biases are updated using these gradients and some type of optimizer, such as stochastic gradient descent or the Adam optimizer.

2.3.2 Deep Q-Networks

One of the first deep reinforcement learning algorithms is Deep Q-Learning, which learns a parameterized state-action value function $Q_\theta(s,a)$ referred to as a Deep Q-Network (DQN) [7]. To stabilize the performance of Deep Q-Learning by removing correlations in observation sequences, an experience replay buffer D is used to store previous state transitions. A target network $\hat{Q}_{\theta^-}(s,a)$ is also used to compute the target value for the loss function, which also stabilizes performance by ensuring the policy does not change too quickly and decorrelates the target values. The weights and biases θ^- are updated periodically to match those of $Q_\theta(s,a)$.

The full algorithm for Deep Q-Learning is provided in Algorithm 3. The algorithm in Reference [7] is modified slightly here to allow for M actors in parallel to interact with the environment, which is common in practice. The algorithm begins by initializing the replay buffer, state-action value function, and a target state-action value function. Then, for each iteration, each actor interacts with the environment by selecting an action following an epsilon-greedy policy and storing the transition in the replay buffer for the set of decision-making intervals I. $|I|$ refers to the maximum number of decison-making intervals. After each step, a gradient descent step is performed on the value function using minibatches sampled from the replay buffer. The target network is updated periodically to match the value function. In the parallel implementation, each actor steps forward once at the same time and the gradient descent step is performed after all agents have stepped forward.

2.3.3 REINFORCE

Policy gradient reinforcement learning algorithms learn a parameterized policy $\pi_\theta(a|s)$, which is a probability distribution conditioned on the state. For certain problems, it is easier to learn a policy directly than it is to learn a value function. A small error in the learned value function may lead to a large error in the policy if the value function is used to select actions. This provides stronger convergence guarantees for policy gradient methods.

Algorithm 3 Deep Q-Learning.

```
1: Initialize replay buffer $D$
2: Initialize state-action value function $Q_\theta(s,a)$ with random weights and biases $\theta$
3: Initialize target state-action value function $\hat{Q}_{\theta^-}(s,a)$ with weights and biases $\theta^- = \theta$
4: for iteration $1 : N$
5:     for $i = 1 : |I|$
6:         for actor $1 : M$
7:             $a_i = \arg\max_a Q_\theta(s_i, a)$ with probability $1-\epsilon$, otherwise select random action
8:             $s_{i+1}, r_i \sim G(s_i, a_i)$
9:             Store transition $(s_i, a_i, r_i, s_{i+1})$ in $D$
10:        Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $D$
11:        $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} \hat{Q}_{\theta^-}(s_{j+1}, a') & \text{otherwise} \end{cases}$
12:        Perform gradient descent step on $(y_j - Q_\theta(s_j, a_j))^2$ wrt $\theta$
13:        Periodically reset $\hat{Q}_{\theta^-} = Q_\theta$
```

Policy gradient algorithms use the policy gradient theorem to compute the gradient of the expected return with respect to the policy parameters θ without knowing how changes to the policy affects the state distributions [32]. The policy gradient theorem provides an expression for the gradient of performance with respect to the policy parameters, and is given by:

$$\nabla_\theta J(\theta) = \sum_s \mu(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a), \tag{2.12}$$

where $\mu(s)$ is a weighting over the state space denoting the probability of being in state s under the policy $\pi_\theta(a|s)$ and $Q_{\pi_\theta}(s,a)$ is the state-action value function for the policy $\pi_\theta(a|s)$. The REINFORCE algorithm is a policy-gradient method that uses the policy gradient theorem to compute the following update for the policy parameters, where α is the step size:

$$\theta \leftarrow \theta + \alpha\gamma^i \left[\sum_{j=i+1}^{|I|} \gamma^{j-i-1} r_j \right] \nabla_\theta \ln \pi_\theta(a_i|s_i) \tag{2.13}$$

To reduce variance, a baseline can be introduced to the policy gradient theorem, which is subtracted from the return. The baseline is typically selected to be a parameterized estimate of the value function, $V_{\theta_v}(s)$, parameterized by the weights and biases θ_v. When introduced into the REINFORCE update, the update becomes:

$$\theta \leftarrow \theta + \alpha\gamma^i \left[\left(\sum_{j=i+1}^{|I|} \gamma^{j-i-1} r_j \right) - V_{\theta_v}(s_i) \right] \nabla_\theta \ln \pi_\theta(a_i|s_i) \tag{2.14}$$

The REINFORCE algorithm is provided in Algorithm 4.

Algorithm 4 REINFORCE with Baseline algorithm.

```
1: Initialize policy $\pi_\theta(s)$ and value function $V_{\theta_v}(s)$ with parameters $\theta$ and $\theta_v$
2: for iteration 1 : $N$
3:     for i = 1 : $|I|$
4:         $a_i \sim \pi_\theta(a_i|s_i)$
5:         $s_{i+1}, r_i \sim G(s_i, a_i)$
6:         Store $s_i, a_i, r_i$
7:         $T = i$
8:         if $s_{i+1}$ is terminal
9:             break
10:    for i = 1 : $T$
11:        $R_i = \sum_{j=i+1}^{T} \gamma^{j-i-1} r_j$
12:        $\theta_v \leftarrow \theta_v + \alpha(R_i - V_{\theta_v}(s_i))\nabla_{\theta_v} V_{\theta_v}(s_i)$
13:        $\theta \leftarrow \theta + \alpha\gamma^i R_i \nabla_\theta \ln \pi_\theta(a_i|s_i)$
```

2.3.4 Advantage Actor-Critic

Actor-critic methods use a learned value function $V_\theta(s)$ to perform the policy update, just as REINFORCE with a baseline does. However, REINFORCE only uses the parameterized value function to estimate the value of the first state for each state transition. This estimate provides a baseline, but does not assess the subsequent return of the action. Actor-critic methods use the value function to estimate the value of the first state as well as the value of the second state. It is for this reason that the actor-critic methods are called actor-critic methods, as they use both an actor (policy) and a critic (value function). The advantage actor-critic algorithm is provided in Algorithm 5. This algorithm is similar to the Asynchronous Advantage Actor-Critic (A3C) algorithm [59]. The A3C algorithm uses multiple actors to collect experience and update the policy and value function asynchronously. A2C does this synchronously.

Algorithm 5 Advantage actor-critic algorithm.

```
1: Initialize policy $\pi_\theta(s)$ and value function $V_{\theta_v}(s)$ with parameters $\theta$ and $\theta_v$
2: for iteration $1:N$
3:     for actor $1:M$
4:         for $i = 1:|I|$
5:             $a_i \sim \pi_\theta(a_i|s_i)$
6:             $s_{i+1}, r_i \sim G(s_i, a_i)$
7:             Store $s_i, a_i, r_i$
8:             if $s_{i+1}$ is terminal
9:                 break
10:        $R = \begin{cases} 0 & \text{if } s_{i+1} \text{ is terminal} \\ V_{\theta_v}(s_{i+1}) & \text{otherwise} \end{cases}$
11:        for $j = |I|:1$
12:            $R = r_j + \gamma R$
13:            Compute advantage estimate $\hat{A}_j = R - V_{\theta_v}(s_j)$
14:            Accumulate gradients wrt $\theta$: $d\theta \leftarrow d\theta + \nabla_\theta \log \pi_\theta(a_j|s_j)\hat{A}_j$
15:            Accumulate gradients wrt $\theta_v$: $d\theta_v \leftarrow d\theta_v + \nabla_{\theta_v} \hat{A}_j^2$
16:        Perform gradient ascent step on $\theta$ and $\theta_v$ using $d\theta$ and $d\theta_v$
```

2.3.5 Proximal Policy Optimization

While DQN, REINFORCE, A2C, and many other methods have advanced the state-of-the-art of reinforcement learning and demonstrated excellent performance on a number of tasks, they are not without their issues. None of these methods are particularly data efficient as each sample is used only once for training. Furthermore, these algorithms are not particularly robust or stable, requiring careful hyperparameter tuning for each task.

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that addresses these issues [60]. To improve sample efficiency, PPO trains on the sampled data for multiple epochs. To improve stability, PPO uses a clipped objective function that ensures the size of the policy update isn't too large. The loss function for PPO is provided by:

$$L_i^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_i \left[L_i^{CLIP}(\theta) - c_1 L_i^{VF}(\theta) + c_2 S[\pi_\theta](s_i) \right], \tag{2.15}$$

where

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_i \left[\min\left(r_i(\theta)\hat{A}_i, \operatorname{clip}\left(r_i(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_i \right) \right], \tag{2.16}$$

$$L^{VF}(\theta) = \hat{\mathbb{E}}_i \left[(V_\theta(s_i) - V)^2 \right], \tag{2.17}$$

$S[\pi_\theta]$ is an entropy bonus, and $r_i(\theta)$ is the probability ratio:

$$r_i(\theta) = \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta^-}(a_i|s_i)} \tag{2.18}$$

θ^- represents the parameters of the policy before the update.

The algorithm for PPO is provided in Algorithm 6. Here we assume that the parameters for the policy and value function are shared.

Algorithm 6 Proximal Policy Optimization algorithm.

```
1: Initialize policy $\pi_\theta(s)$ and value function $V_\theta(s)$ with parameters $\theta$
2: Initialize policy $\pi_{\theta^-}(s)$ and value function $V_{\theta^-}(s)$ with parameters $\theta^-$
3: for iteration $1 : N$
4:     for actor $1 : M$
5:         for $i = 1 : |I|$
6:             $a_i \sim \pi_{\theta^-}(a_i|s_i)$
7:             $s_{i+1}, r_i \sim G(s_i, a_i)$
8:             Store $s_i, a_i, r_i$
9:         compute advantage estimates $\hat{A}_1 \cdots \hat{A}_{|I|}$
10:    optimize $L(\theta)$ wrt $\theta$, with $K$ epochs and batch size $\leq |I|$
11:    $\theta^- \leftarrow \theta$
```

2.3.6 Shielded PPO

While PPO is a robust and stable algorithm capable of computing high performing policies for a number of RL problems, PPO does not guarantee that unsafe actions will not be taken and that resource constraint violations will not occur. In fact, none of the aforementioned DRL algorithms do. Shielded deep reinforcement learning offers a solution to this problem by using a linear temporal logic specification to monitor the MDP state and the actions output by the policy, overriding unsafe actions if they violate the specification [53]. Harris and Schaub utilize a safety shield within PPO for spacecraft planning and scheduling, which is shown to improve the speed of convergence and guarantee that resource constraint violations do not occur. This algorithm is

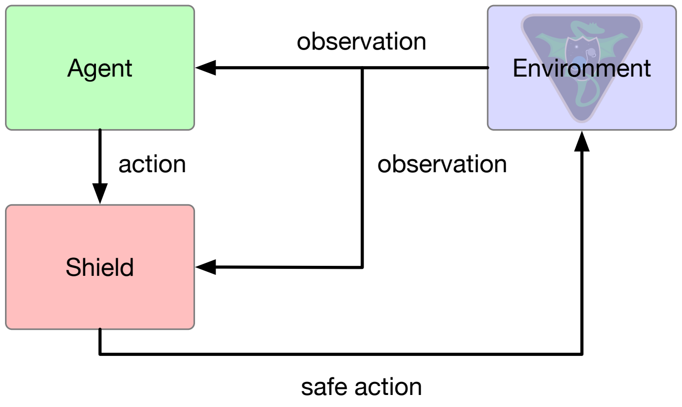


Figure 2.4: Shielded agent-environment interface.

referred to as SPPO [52, 61]. A diagram of the safety shield augmented agent-environment interface is provided in Figure 2.4.

2.3.7 Conclusion

Reinforcement learning is a powerful collection of problem formulations and solution methods for planning problems. This section detailed how reinforcement learning problems are formulated as Markov decision processes and solved with a variety of solution methods. Tabular methods are very powerful methods that can solve for optimal policies and value functions, but are typically reserved for smaller problems with discrete state and action spaces. Deep reinforcement learning methods parameterize the value function or policy with an artificial neural network, allowing these solution methods to only learn from a subset of the state space and extrapolate experience. DRL methods are extremely powerful and have been shown to solve complex problems with high-dimensional state and action spaces.

Chapter 3

Earth-Observing Satellite Scheduling Problem Formulation

3.1 Introduction

In the Earth-observing satellite scheduling problem, one or more spacecraft in orbit about the Earth attempt to maximize the amount of science collected and/or downlinked while avoiding resource constraint violations. Many variations of this problem exist. Some variations consider only one satellite, while other variations consider multiple satellites. Science objectives may include area coverage objectives, ground target imaging objectives specific to a geographic location, or both simultaneously. Furthermore, a variety of spacecraft resources such as battery charge, data buffer storage capacity, or reaction wheel speeds may be considered in the problem formulation. The agile EOS scheduling problem is one of the most common in the literature, especially in recent years as satellites with three-axis attitude control capabilities have become more prominent. The objective of the agile EOS scheduling problem is to maximize the weighted sum of imaging targets collected and downlinked while avoiding resource constraint violations. The satellite is referred to as "agile" because its three-axis attitude control capabilities enable attitude maneuvers about any axis, as opposed to only pitch or roll maneuvers. This chapter only considers the single satellite agile Earth-observing (SSAEO) scheduling problem, as opposed to the multi-satellite agile Earth-observing scheduling problem.

In general, authors utilize simple models of the Earth-observing satellite scheduling problem and do not adequately address resource constraints and their impact on the planning problem. In the surveyed works, only two authors take into account power constraints [52, 54], two authors take into account data buffer constraints [49, 54], and one author takes into account momentum management [52]. Three authors take into account none of these resource constraints [50, 51, 62]. Furthermore, many of these works rely on simplifying assumptions of the problem dynamics, with the exception of Harris and Schaub who consider the full non-linear dynamics of the EOS scheduling problem and leverage the black box optimization capabilities of DRL [52]. Eddy and Kochenderfer make linearity assumptions regarding the power and data dynamics. While linear data dynamics are sufficient, linear power dynamics may not be appropriate due to the dependency of power generation on the sun's incidence angle with respect to the solar panels. Eddy and Kochenderfer also rely on an agility constraint to assess whether transitions between pointing configurations are feasible [54]. The assumptions made by Eddy and Kochenderfer allow the authors to make fair comparisons between methods like Monte Carlo tree search, graph search, and mixed integer linear programming. Other authors use even more simple models of the problem. The target sets are usually pre-processed from actual orbital dynamics, but the actions and subsequent problem dynamics are oversimplified. Haijiao et al. only evaluate whether the next observation task is accepted or rejected [49]. Zhao et al. formulate a two-phase combinatorial reinforcement learning problem [50]. In phase 1, a recurrent neural network architecture based on reinforcement learning is used to sequence a series of acquisitions. In phase 2, reinforcement learning is used to determine the start time of the observation window of the next acquisition. He et al. investigate a problem in which an observation task is added to a queue of observation tasks for execution [51]. Again, these models of the EOS scheduling problem abstract away many of the complexities of the real problem.

To address these shortcomings in the literature, this chapter formulates an SSAEO scheduling problem that considers battery, data buffer, and reaction wheel resource constraints. Furthermore, the problem is modeled using a high-fidelity astrodynamics simulation framework. The use of this simulator allows for the nonlinear spacecraft dynamics, power and data subsystems, and real flight software models to be used within the problem formulation. This work is published in References [63] and [64].

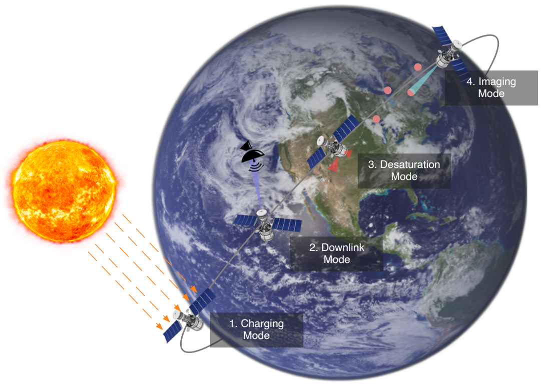


Figure 3.1: Single satellite agile Earth-observing scheduling problem.

3.2 Single Satellite Agile Earth-Observing Scheduling Problem

In the SSAEO scheduling problem, the satellite enters into different operational modes to achieve its objective over the planning horizon. The planning horizon is the total amount of time considered for operations and is divided into a set of equal length decision-making intervals. This set is referred to as I. At the beginning of each decision-making interval i, the satellite either stays in the same operational mode or enters a new operational mode. The operational modes abstract high-level behaviors whose low-level behavior is dictated by the interaction between the different subsystems of the satellite and the operational environment. The modes also dictate the attitude reference and state of each satellite subsystem. This mode-based planning approach breaks the complex, continuous behavior of the satellite and the scheduling problem into a set of discrete actions to make the planning problem tractable [52]. Operational modes can include science modes, momentum management modes, charging modes, or downlink modes. A diagram of the EOS scheduling problem is provided in Figure 3.1.

The satellite has a set of targets available for imaging, ordered by the time the spacecraft has access to the targets. This set of targets is referred to as T. An example of the set T may be found in Figure 3.2. In this example, the ground targets are quite close to the ground track of the satellite. A subset of T that includes the next J upcoming targets is also formulated, and this set is referred to as U. The set D includes all targets that have already been passed or imaged by the

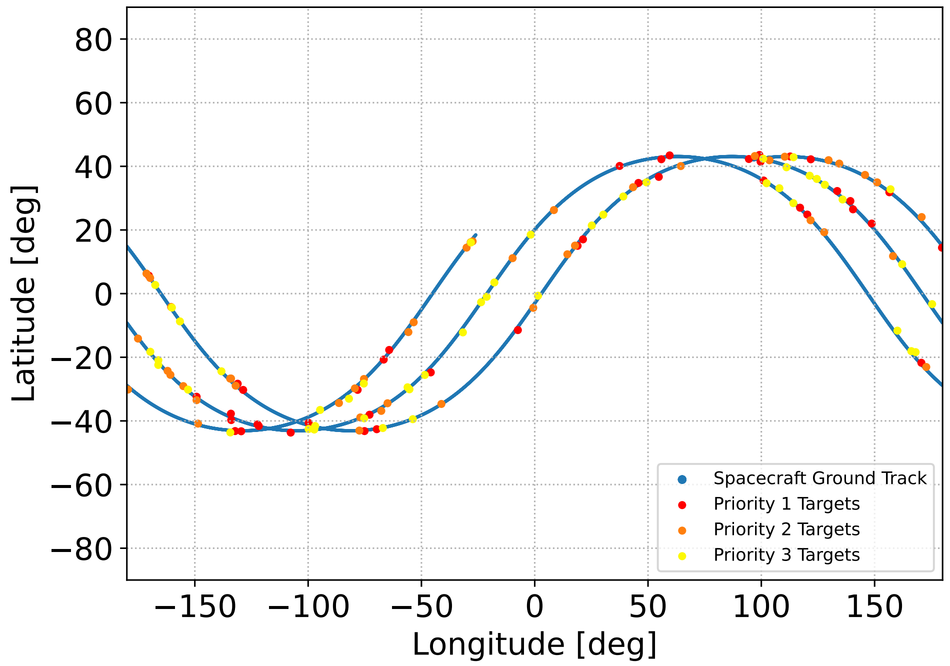


Figure 3.2: Target distributions.

satellite.

$$U = \{c_j \in (T - D) \mid \forall\, j \in [1, J]\} \tag{3.1}$$

The sets U and D are updated at the end of each decision-making interval based on the actions of the satellite and which target have been passed over already.

The satellite has three operational modes available for resource management and J operational modes available for science activities. The resource management modes include the charging mode, desaturation mode, and downlink mode. In the charging mode, the satellite turns off its instrument and transmitter and points its solar panels at the sun to charge its batteries. The desaturation mode is the same as the charging mode, but the satellite uses its thrusters to perform burns that remove momentum from the reaction wheels. In the downlink mode, the satellite points in the nadir direction and downlinks its science data when a ground station is in view. A ground station is within view when the satellite is within the elevation and range requirements of the ground station. A diagram of these requirements is provided in Figure 3.3. The satellite's position is first transformed into the topocentric horizon coordinate system, i.e. the "South-East-Zenith"

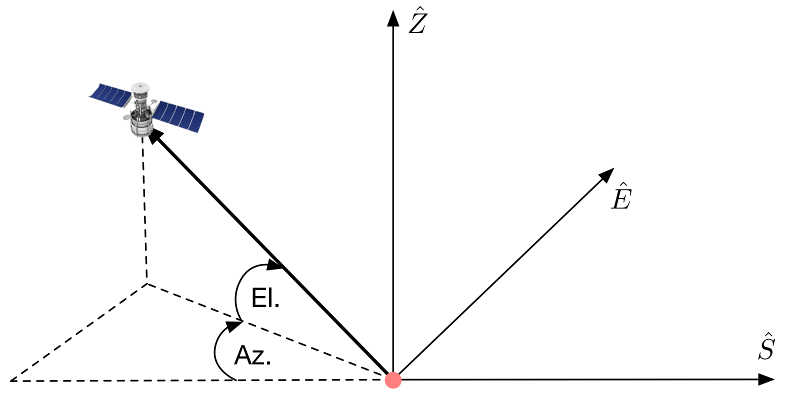


Figure 3.3: Elevation and range requirements.

or SEZ frame, and then the range and elevation are computed.

The science-related operational modes of the SSAEO scheduling problem deal with target imaging. In the target imaging mode, the satellite turns off its transmitter, turns on its instrument, and points the boresight of its instrument at the selected ground target in the set U. An image of the ground target is collected once the elevation, range, and an additional attitude error requirement are met. The collected image is then stored in the data buffer, where it is available to downlink in the future.

3.3 Markov Decision Process Formulation

This section formalizes the SSAEO scheduling problem as a Markov decision process. The state space, action space, and reward function are all explained in detail. The generative transition function is explained in the next section.

3.3.1 State Space

One of the most challenging aspects of formulating a real-world problem as an MDP is designing the state space. The state space must contain all information relevant to the decision-making problem that ensures the Markov property is satisfied. For the EOS scheduling problem, this is difficult to completely satisfy due to the complexity of the problem, which requires a massive state space to fully capture. Therefore, the state space is selected to contain the information that is most relevant to the decision-making problem.

The state space for the agile EOS scheduling problem includes the state of the satellite, the state of the ground stations, and the state of the ground targets. It may formally be defined as:

$$\mathcal{S} = \mathcal{S}_{\text{sat}} \times \mathcal{S}_{\text{ground stations}} \times \mathcal{S}_{\text{targets}}, \tag{3.2}$$

where state space of the satellite includes the position, velocity, attitude, wheel speeds, power states, buffer states, and eclipse states. It is defined as

$$\mathcal{S}_{\text{sat}} = \mathcal{S}_{\text{pos}} \times \mathcal{S}_{\text{vel}} \times \mathcal{S}_{\text{att}} \times \mathcal{S}_{\text{wheels}} \times \mathcal{S}_{\text{power}} \times \mathcal{S}_{\text{buffer}} \times \mathcal{S}_{\text{eclipse}}, \tag{3.3}$$

Finally, for this variant of the SSAEO scheduling problem, the state space of the imaging targets includes the positions and priorities of the targets and is defined as:

$$\mathcal{S}_{\text{targets}} = \mathcal{S}_{\text{pos}_1} \times \mathcal{S}_{\text{priority}_1} \times \cdots \times \mathcal{S}_{\text{pos}_{|T|}} \times \mathcal{S}_{\text{priority}_{|T|}} \tag{3.4}$$

The state returned to the decision-making agent at decision interval i is defined as $s_i \in S$:

$$\begin{aligned} s_i = ({}^{\mathcal{E}}\mathbf{r},\ {}^{\mathcal{E}}\mathbf{v},\ \|\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}\|,\ \|{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}\|, \boldsymbol{\Omega},\ \text{battery}, \cdots \\ \cdots \text{eclipse}, \text{buffer}, {}^{\mathcal{H}}\boldsymbol{r}_1,\ p_1,\ \cdots {}^{\mathcal{H}}\boldsymbol{r}_j,\ p_j). \end{aligned} \tag{3.5}$$

The position and velocity of the spacecraft included in the state return are expressed in the Earth-centered, Earth-fixed (ECEF) coordinate system, which is denoted by the left superscript $\mathcal{E}$. Because these are expressed in the ECEF frame, they can be used to correlate positions over ground stations to high value states in the solved MDP. Information regarding the attitude of the satellite is provided with two separate state variables. The attitude of the satellite is provided as the magnitude of the modified Rodriguez parameters (MRP) $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$, which is rotation from the reference frame to the body frame. The magnitude of the angular velocity of the satellite ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$ is provided in the body frame. These two variables provide attitude error and the rotation rate of the satellite, with the latter being important for momentum management. Regarding momentum management, each of the reaction wheel speeds is given by $\boldsymbol{\Omega}$, which provides information regarding the wheels speeds and when a desaturation maneuver may be required. Information regarding power management is included using the battery and eclipse state variables. The battery variable is the current charge of the battery, and the eclipse variable is a binary variable that indicates whether the satellite is currently in eclipse. Similar to the battery variable, a variable for the buffer storage level is also provided, which provides information on when a downlink may be required.

Finally, state information is included for each target in the set of upcoming targets U. The position of each target expressed in the spacecraft's Hill frame ${}^{\mathcal{H}}\boldsymbol{r}_j$ is included in the state return. The priority p_j of each target is included as well. The priority of each target is a value between 1 and 3, where 1 is the highest priority and 3 is the lowest priority. The priorities are randomly selected during the target generation step. The expression of the positions of the targets in the satellite Hill frame is selected to provide a convenient state representation for the purposes of function approximation. A diagram of the Hill frame is provided in Figure 3.4. Furthermore, only the targets in the set of upcoming targets U are included in the state. While this does add some observability challenges to the problem, there is a value for $|U|$ that will render this impact negligible because the added information of more targets will only marginally improve observability while increasing problem complexity and required training time. If the agent has observability over every target and their priorities, it would have perfect information and the ability to compute the value exactly. In future chapters, the size of U is explored to determine when the agent has enough information to extract the maximum possible reward from the environment.

As referenced before, each state return is normalized to ensure the problem is numerically well-conditioned during function approximation. The normalization constants for each state returned are provided in Table 3.1. States without any normalization listed are not normalized as their values typically fall within the range of [-1, 1].

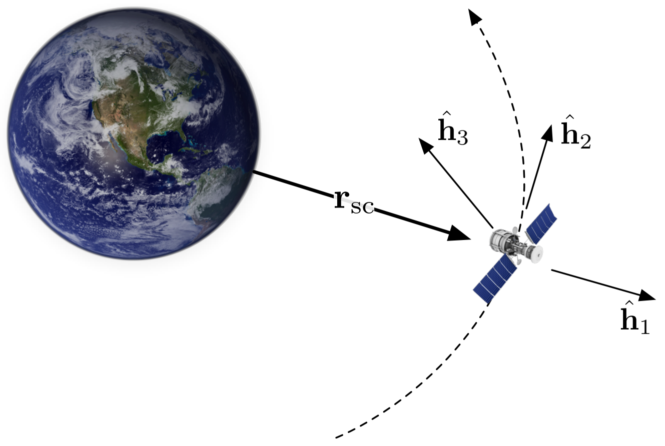


Figure 3.4: Diagram of the Hill coordinate system.

Table 3.1: State normalization.

State	Normalization
${}^{\mathcal{E}}\boldsymbol{r}$	Radius of Earth
${}^{\mathcal{E}}\boldsymbol{v}$	Velocity of circular orbit at Earth's surface
${}^{\mathcal{H}}\boldsymbol{r}_j$	Radius of Earth
p_j	p_j^2
$\|\|\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}\|\|$	–
$\|\|{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}\|\|$	–
$\boldsymbol{\Omega}$	Maximum wheel speeds
Battery charge	Maximum battery capacity
Stored data	Maximum storage capacity
Eclipse indicator	–

3.3.2 Action Space

An action space $\mathcal{A}$ is constructed for the SSAEO scheduling problem that allows the decision-making agent to collect and downlink science data as well as manage its resources:

$$\mathcal{A} = \{\text{Charge, Downlink, Desaturate, Image } c_1, \cdots \text{Image } c_j\}. \tag{3.6}$$

At each decision-making interval, the satellite turns on or off certain attitude references, instruments, and transmitters for the duration of the decision-making interval. The continuous behavior of the satellite system is thus abstracted using discrete actions, or modes. The action space, as well as a description for each mode, is provided below:

(1) **Charge:** The satellite points its solar panels at the sun, turning off all instruments and transmitters to recharge the batteries.

(2) **Desaturate:** The satellite points its solar panels at the sun, turning off all instruments and transmitters. Momentum is mapped to thrust commands, which the thrusters execute.

(3) **Downlink:** The satellite points the transmitter at the Earth. The transmitter is turned on and data is downlinked if and when a ground station is available.

$(3+1)$ **Image target** $c_1 \in U$

$\vdots$

$(3+J)$ **Image target** $c_j \in U$**:** The satellite points the instrument at the target, taking an image of the target when access requirements are met. The data is stored on-board the satellite.

3.3.3 Reward Function

A piecewise reward function, $R(s_i, a_i, s_{i+1})$, is developed for the SSAEO scheduling problem to ensure that science data is collected and downlinked and that resource constraint violations are avoided. The reward function is provided in Equation 3.7.

$$R(s_i, a_i, s_{i+1}) = \begin{cases} -10 & \text{if failure} \\ \sum_j^{|\boldsymbol{T}|} H(d_j) & \text{if } \neg\text{failure} \wedge a_i \text{ is downlink} \\ 0.1H(w_j) & \text{if } \neg\text{failure} \wedge a_i \text{ is image } c_j \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

The failure condition is checked first. Failure occurs if the battery is drained to zero charge, the data buffer is overfilled, or the reaction wheel speeds exceed the maximum speed. The failure condition is provided in Equation 3.8. If a failure condition occurs, a reward penalty of -10 is returned. This penalty is sized to incur a relatively large penalty if the decision-making agent fails. However, this dissertation also investigates different reward penalties relative to the positive reward.

$$\text{failure if battery} = 0,\ \text{any}(\boldsymbol{\Omega}/\Omega_{\max} \geq 1),\ \text{or buffer} \geq 1 \tag{3.8}$$

If a failure does not occur and the downlink action is taken, the local target list is checked to determine if a target was downlinked for the first time or not. The function that performs this check for imaging and downlink is provided in Equation 3.9. If a target is imaged or downlinked for the first time, then 1 divided by the target priority is returned.

$$H(x_j) = (1/p_j) \text{ if } \neg x_{j_i} \ \wedge \ x_{j_{i+1}} \tag{3.9}$$

A summation over Equation 3.9 is performed and normalized by the maximum number of decision-making intervals, $|I|$, to ensure the reward contribution from downlinking targets does not exceed 1.

If no failure occurs and an imaging action is taken, Equation 3.9 is applied to that ground target. This component of the reward is sized such that the maximum amount of reward from imaging does not exceed 0.1. Without the small reward bonus for imaging, the sparsity of the reward can impede learning. Furthermore, the decision-making agent needs a reward incentive to image when all downlink windows have been passed but the end of the planning horizon has not yet arrived.

3.4 Basilisk Gymnasium Interface

The generative transition function of the SSAEO scheduling problem $s_{i+1}, r_i \sim G(s_i, a_i)$ is modeled using a high-fidelity astrodynamics simulation framework, Basilisk[1] [65]. Basilisk implements simulation and flight software code in C/C++, but provides a Python interface for scripting. The Basilisk simulation is wrapped within a Gymnasium[2] environment, which provides a standard interface that allows reinforcement learning libraries to interact with the simulation. The

[1] https://hanspeterschaub.info/basilisk/index.html
[2] https://gymnasium.farama.org/

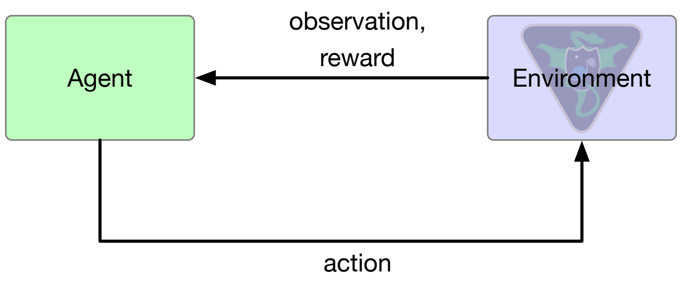


Figure 3.5: Agent-environment interface.

agent passes actions to the Gymnasium environment, which turns certain Basilisk modules on or off. The simulation is integrated forwards in time for six minutes at a one-second integration time step. The environment constructs the observation and reward, which is returned to the decision-making agent. This process is demonstrated in Figure 3.5.

A complete diagram of the associated Basilisk modules may be found in Figure 3.6. Each module in the diagram represents a distinct, modularized block of code that receives inputs from other modules, performs computations, and sends outputs to modules subscribed to its messages. These connections are not shown in the diagram to maintain clarity of the figure. Each module belongs to a task, which may be turned on or off depending on the operational mode. The dynamics, environment, and Spice tasks are always on, but the flight software tasks depend on the operational mode. There is no requirement on which tasks belong to the environment and dynamics tasks, as long as they are added to the simulation in the correct order.

The Basilisk simulation includes a full attitude control system to simulate a representative spacecraft mission where systems are coupled to the physical attitude dynamics. Several pointing tasks are created, each of which contain a different location pointing or Hill pointing object that computes an attitude reference for the corresponding celestial object or ground location. The selected attitude reference is passed to the MRP feedback control law, which sends motor torque commands to reaction wheels to change the dynamics of the spacecraft through the use of the reaction wheel motor torque module. Both of these modules belong to the MRP feedback task, which is always on. The reaction wheels are modeled after the Honeywell HR16 reaction wheels. In

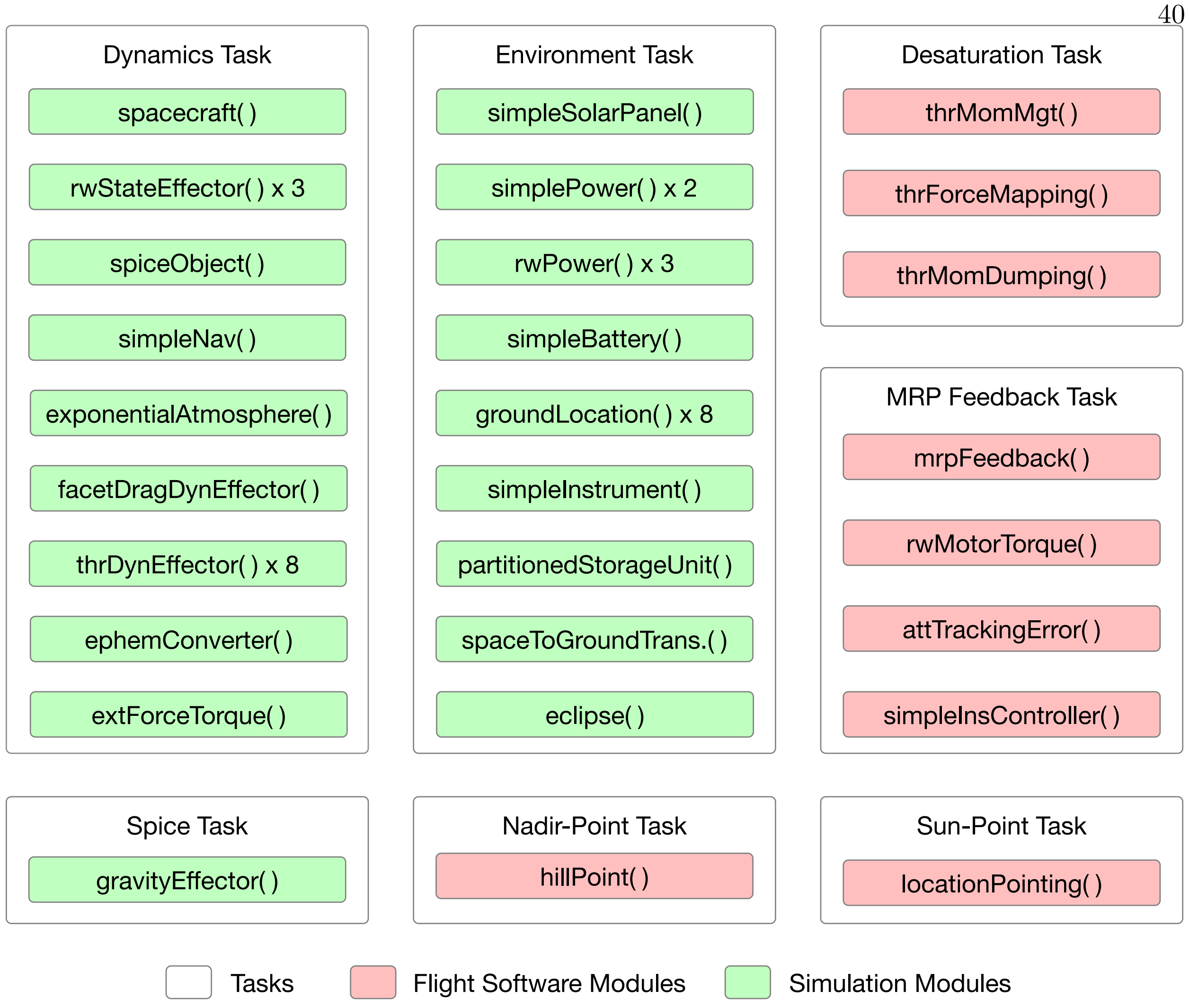


Figure 3.6: SSAEO Basilisk simulation architecture.

the desaturation task, the thruster momentum management, thruster force mapping, and thruster momentum dumping modules work together to map reaction wheel momentum to thruster on-time commands. The thrusters are modeled after the Moog Monarc-1 thrusters. Attitude perturbations are incorporated through the use of random external disturbance torques to build up momentum in the reaction wheels. Orbital perturbations like multi-body gravity effects (including Earth, sun, and the moon) and Earth J_2 perturbations are also implemented. A summary of the dynamics models and the relevant states they impact are provided in Table 3.2, and the parameters of the spacecraft may be found in Table 3.3.

Table 3.2: SSAEO Basilisk tasks and models.

	Operational Modes			
Basilisk Tasks & Models	Charge	Downlink	Desaturate	Image
Location Point Task	Disabled	Disabled	Disabled	Enabled
Nadir Point Task	Disabled	Enabled	Disabled	Disabled
Sun-Point Task	Disabled	Enabled	Disabled	Disabled
MRP Control Task	Enabled	Enabled	Enabled	Enabled
RW Desat Task	Disabled	Disabled	Enabled	Disabled
Instrument Power Model	Off	Off	Off	On
Instrument Data Model	Off	Off	Off	On
Transmitter Power Model	Off	On	Off	Off
Transmitter Data Model	Off	On	Off	Off

A power system is modeled that includes solar panels, an instrument power model, a transmitter power model, reaction wheel power models, and a battery. The solar panels generate power based on their efficiency, surface area, and the incidence angle of the sun. The power generated by the solar panels is stored within a battery, which has a maximum capacity. The battery is used to power the instrument, transmitter, and reaction wheels. The instrument and transmitter power models are used to determine the power draw of the instrument and transmitter based on the current operational mode. The reaction wheel power models are used to determine the power draw of the reaction wheels based on their current speed and commanded torque. The power system is modeled in Basilisk, and the parameters of the power system may be found in Table 3.3.

An on-board data system is also modeled in Basilisk. The on-board data system includes an instrument, a transmitter, and a data buffer. The parameters associated with these components may also be found in Table 3.3. The ground segment of the data system includes several ground stations located on the surface of the Earth. The ground stations are selected from NASA's Near Space Network [66], and their parameters may be found in Table 3.4. The ground stations are selected to provide an adequate distribution on the Earth's surface so that the satellite has at least one downlink opportunity over its planning horizon.

Table 3.5 shows the distributions that the initial conditions are drawn from each time a new SSAEO scheduling environment is initialized. The eccentricity, inclination, argument of periapsis,

Table 3.3: SSAEO spacecraft parameters.

General Spacecraft Parameters	Value
Mass	330 kg
Dimensions	1.38 x 1.04 x 1.58 m
Inertia	diag(82.1, 98.4, 121) kg m^2
Power System	
Solar Panel Area	1.0 m^2
Solar Panel Efficiency	0.20
Instrument Power Draw	30 W
Transmitter Power Draw	15 W
Battery Capacity	80 Whr
RW Base Power Draw	0.4 W
RW Efficiency	0.5
Attitude Control System	
Max Wheel Speeds	3000 RPM
Max Wheel Torque	0.2 Nm
Max Thrust	0.9 N
Thruster Min On Time	0.02 s
Data & Communications System	
Data Buffer Storage Capacity	20 Images
Instrument Baud Rate	1 Image/second
Transmitter Baud Rate	1 Image/second

Table 3.4: SSAEO ground station parameters.

Location	Latitude	Longitude	Elevation (m)	Min. El. Angle
Boulder, CO (USA)	40.015° N	105.27° W	1600 m	10°
Ka Lae, HI (USA)	19.897° N	155.58° W	9.0 m	10°
Merritt Island, FL (USA)	28.318° N	80.666° W	0.91 m	10°
Singapore	1.3521° N	103.82° E	15 m	10°
Weilheim, Germany	47.841° N	11.142° E	560 m	10°
Santiago, Chile	33.449° S	70.669° W	570 m	10°
Dongara, Australia	29.245° S	114.93° E	34 m	10°

and true anomaly are drawn from fairly large distributions such that the agent experiences any possible low-Earth orbit with the given semi-major axis. The semi-major axis is initialized to 6,871 km in all cases. While the semi-major axis is fixed, some variation in altitude is inherent due to the eccentricity and the previously described perturbations. The longitude of ascending node is also drawn from a small range. This allows for experiments to determine how well the decision-making agents generalize to ranges outside of the training distributions. Also provided in Table 3.5 are the

Table 3.5: SSAEO simulation parameters.

Orbit Parameters	Value
Semi-Major Axis, a	6871 km
Eccentricity, e	$\mathcal{U}$[0, 0.01]
Inclination, i	$\mathcal{U}$[40, 60] deg
Long. of Ascend. Node, Ω	$\mathcal{U}$[0, 20] deg
Arg. of Periapsis, ω	$\mathcal{U}$[0, 360] deg
True Anomaly, f	$\mathcal{U}$[0, 360] deg
Spacecraft Parameters	
Disturbance Torque, $\boldsymbol{\tau}_{\text{ext}}$	2×10^{-4} Nm
Attitude Initialization, $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$	$\mathcal{U}$[0, 1.0] rad
Rate Initialization, ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$	$\mathcal{U}$[-1e-05, 1e-05] rad/s
Reaction Wheel Speeds	$\mathcal{U}$[-1500, 1500] RPM
Initial Battery Charge	$\mathcal{U}$[30, 70] Whr
Planning Horizon	
Length	270 minutes
Decision-Making Interval Length	6 minutes
Integration Time Step	1 second

initial conditions for the spacecraft attitude and rate, reaction wheel speeds, and initial battery charge. A planning horizon of 270 minutes (approximately three orbits) is selected to balance computation time with a challenging operational scenario. Furthermore, each decision-making interval lasts for a total of six minutes. Six minutes is selected as the length of the decision-making interval due to the time it takes for the attitude control system to converge to the new reference if a change is made. Six minutes also provides adequate time for a desaturation maneuver to take place. Finally, the dynamics are integrated with a 1-second time step.

3.5 Conclusion

This chapter presented a problem formulation for the single satellite agile Earth-observing scheduling problem that adequately addresses resource constraints and leverages a high-fidelity astrodynamics simulation framework to model the problem. The problem was described and formulated as a Markov decision process, which includes the definition of the state space, action space, reward function, and transition function. The parameters of the high-fidelity astrodynamics simulations were also presented. This problem formulation and simulator will be used in the following chapters to develop and evaluate a variety of RL-based scheduling algorithms.

Chapter 4

MCTS-Train

4.1 Introduction

Monte Carlo tree search, introduced in Chapter 2, is an online search algorithm that may be used to solve sequential decision-making problems [55, 57]. In this chapter, MCTS-Train is introduced and deployed on two versions of the EOS scheduling problem. MCTS-Train is a training pipeline that uses MCTS to generate estimates of the state-action value function and supervised learning to produce a parameterized state-action value function $Q_\theta(s, a)$. MCTS-Train takes inspiration from the AlphaGo algorithm, which uses neural networks in Monte Carlo tree search to either guide rollouts or replace them entirely [1, 67]. The reason for this is that exhaustive search in the game of Go is impossible because of the number of trajectories through the environment (roughly 35 legal moves per board state assuming a game length of 80 moves, equating to $\sim 35^{80}$ trajectories). The AlphaGo Zero algorithm uses a neural network within MCTS to estimate the state-action value function in place of executing rollouts [1]. This is an improvement over AlphaGo Fan, which uses a neural network to execute the Monte Carlo rollouts [67]. One of the major reasons that this is done is because it is difficult to craft a good rollout policy for the game of Go. Even if a good rollout policy is crafted, the performance of MCTS may be limited by the quality of the rollout policy for a given number of simulations.

While AlphaGo Zero achieves superhuman performance in the game of Go, this comes at a computational cost. First, the algorithm requires approximately 70 hours of wall clock time to produce a policy, with the search process, function approximation, and evaluation occurring asynchronously in parallel. Second, during the search step of Monte Carlo tree search, AlphaGo Zero executes approximately 1,600 simulations in 0.4 seconds. This is likely due to the parallel implementation and the relatively low computational cost of evaluating board states and rules. In comparison, the SSAEO scheduling problem presented in Chapter 3 takes 0.11 seconds ($\mu = 0.11, \sigma^2 = 0.0052$) on a 2.8 GHz CPU to execute a single step. This is due to the fact that the EOS scheduling problem requires equations of motion to be integrated and complicated observations to be constructed. It's worth noting that in comparison to the $\sim 35^{80}$ trajectories in Go, the SSAEO scheduling problem has $(3 + |U|)^{45}$ trajectories, where $|U|$ is typically between one and five. While this reduction in the number of total trajectories reduces the number of simulations required to build an adequate search tree, it's difficult to know for sure how this will manifest until the entire training architecture is developed and the algorithm is benchmarked. Furthermore, because the SSAEO scheduling problem is implemented using a high-fidelity simulation that cannot be saved off in memory and restarted from some arbitrary state, the entire simulation must be rewound by re-executing the action history at each simulation step. In contrast, a board state in the game of Go is trivial to re-initialize.

Finally, it's worth considering whether the power of AlphaGo Zero's policy improvement operation is really necessary for the EOS scheduling problems. While it may be difficult to construct a rollout policy for the game of Go that will achieve good performance, the EOS scheduling problem is well understood and a simple rollout policy that respects all safety constraints and collects and downlinks science data can be constructed with relative ease. If the rollout policy is performant enough, only a few search steps in MCTS can result in large performance improvements and a near-optimal policy. The state-action value estimates generated with MCTS can then be regressed over to approximate this near-optimal policy.

This chapter describes the MCTS-Train pipeline in detail and present two case studies on how the algorithm can be used to solve and even parameterize EOS scheduling problems. Two EOS scheduling problems are evaluated: a simple EOS scheduling problem [68, 69] and the agile EOS scheduling problem presented in Chapter 3 (referred to as the SSAEO scheduling problem)

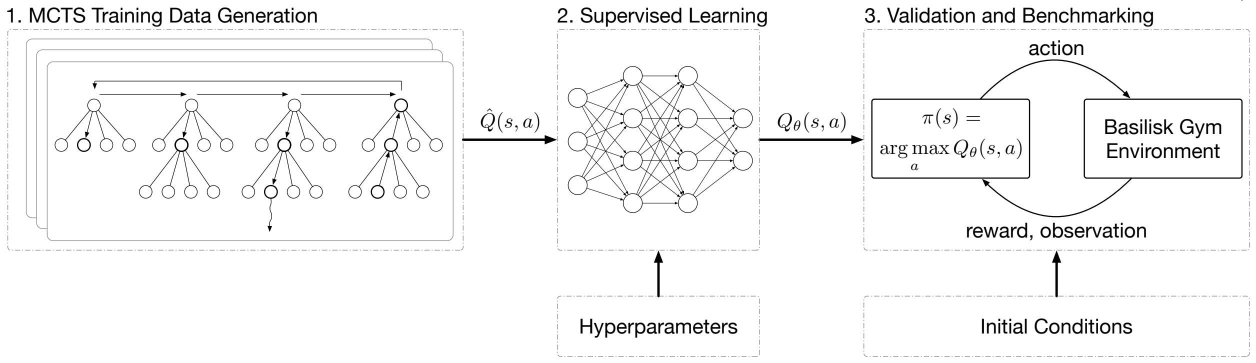


Figure 4.1: MCTS-Train pipeline. MCTS is used to generate thousands of estimates of the state-action value function. Various artificial neural networks are used to regress over the training data, and each neural network is then validated and benchmarked in the environment.

[63, 64]. Each problem considers the same resources, but the science objectives of each problem are quite different. The simple EOS scheduling problem only requires that the satellite points its instrument in the nadir direction to collect data; no imaging targets are considered.

4.2 MCTS-Train

4.2.1 Overview

The full algorithm for MCTS-train may be found in Algorithm 7, and a corresponding figure of the training pipeline may be found in Figure 4.1. The pipeline can be broken down into three distinct steps, each of which is described here in detail. In step one, the training data set is generated using hundreds or thousands of solutions found by MCTS. In step two, supervised learning is applied to generate a parameterized state-action value function. In step three, the trained policies are executed in the environment and their performance is benchmarked.

4.2.2 Training Data Generation

4.2.2.1 Overview

MCTS-Train first generates a training data set $\mathbf{Q}$ of state-action value estimates $\hat{Q}(s,a)$ by solving the EOS scheduling problem for hundreds or thousands of unique initial conditions using

Algorithm 7 MCTS-Train algorithm.

```
1: Initialize set of training data, $\mathbf{Q}$
2: for iteration $1 : N$
3:     for iteration $1 : |I|$
4:         $a_i$ = MCTS.selectAction($s_i$)
5:         $s_{i+1}, r_i \sim G(s_i, a_i)$
6:     Add $\hat{Q}(s_i, a_i)$ from MCTS to $\mathbf{Q}$

7: Initialize set of hyperparameters
8: Initialize empty set of networks, $\mathbf{Q}_\theta$
9: for hp $\in$ hyperparameters
10:     initialize $Q_\theta$ with hp
11:     $Q_\theta$.train(hp)
12:     $\mathbf{Q}_\theta \cup \{Q_\theta\}$

13: for $Q_\theta \in \mathbf{Q}_\theta$
14:     reward_sum = 0
15:     for iteration $1 : |I|$
16:         $a_i = \arg\max_{a_i} Q_\theta(s_i, a_i)$
17:         $s_{i+1}, r_i \sim G(s_i, a_i)$
18:         reward_sum += $r_i$
19:     Save performance metrics
```

Monte Carlo tree search. For each unique initial condition, MCTS builds a search tree by simulating hundreds of interactions with the environment. A state-action value estimate is maintained as MCTS steps through the environment, which is exploited to select the next best action after a pre-determined number of simulations have been executed.

4.2.2.2 Rollout Policy

The full algorithm for MCTS is described in detail in Chapter 2.2.3 (Figure 2.2 and Algorithm 2). The selection and expansion steps are described in detail there. After selection and expansion, a rollout policy is executed to the desired depth of search. The rollout policy can take many forms. A random rollout policy, $\pi_{\text{rand}}(s)$, randomly executes actions. A heuristic rollout policy executes actions according to a pre-defined heuristic. In this work, both types of rollout policies are compared. The random rollout policy simply selects from a uniform distribution over the set of available actions.

To create the rollout policy, a safety MDP is derived as described by Harris and Schaub [52, 61]. The safety MDP discretizes the state space to reduce dimensionality to several safety states. The state space of the safety MDP is defined as follows:

$$\mathcal{S}_{\text{safety}} = \mathcal{S}_{\text{tumbling}} \times \mathcal{S}_{\text{saturated}} \times \mathcal{S}_{\text{low power}} \times \mathcal{S}_{\text{buffer overflow}} \tag{4.1}$$

The safety states take a boolean value of 0 or 1 based on whether the relevant resource state variables are above or below a safety limit. When the safety MDP achieves a nominal state ($s_i = (0, 0, 0, 0)$), meaning that any action can be safely taken, or if the safety MDP indicates the satellite is tumbling only ($s_i = (1, 0, 0, 0)$), an imaging mode is selected. In the simple EOS scheduling problem, the satellite only has one imaging mode. However, the SSAEO scheduling problem has many imaging targets that may be selected. When MCTS is applied to the SSAEO scheduling problem, the closest target in $|U|$ is selected for imaging. For either problem, if the safety MDP is in any other state, a resource management action is selected. The safety states are defined as follows:

$$s_{\text{tumbling}} = \|{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}\| \geq \text{Rotation Rate Limit} \tag{4.2}$$

$$s_{\text{saturated}} = \|\boldsymbol{\Omega}\|/\Omega_{\max} \geq \text{Wheel Limit} \tag{4.3}$$

$$s_{\text{low power}} = \text{battery/max storage} \leq \text{Battery Limit} \tag{4.4}$$

$$s_{\text{buffer overflow}} = \text{buffer/max storage} \geq \text{Buffer Limit} \tag{4.5}$$

The limits of the safety states are problem-specific and may be found in Table 4.1. The differences in limits are largely due to the simple EOS scheduling problem having a larger maximum reaction wheel speed and the instrument collecting far more data per decision-making interval than the SSAEO scheduling problem.

Table 4.1: State limits for shield.

State	Simple EOS	Agile EOS
Rotation Rate Limit	1e-2 rad/s	1e-2 rad/s
Wheel Limit	0.64	0.6
Battery Limit	0.5	0.25
Buffer Limit	0.8	0.95

To ensure a safe action is taken if the satellite is in an unsafe state, a policy is generated for the safety MDP that guarantees a resource constraint failure does not occur if the safe action is taken. This policy is referred to as $\pi_{\text{safe}}(s)$. The MDP safety limits and policy actions are both hand tuned and benchmarked to ensure no failures occur. The policy is provided in Table 4.2. If the imaging action is selected, access to the ground stations is also checked, and if any ground stations are accessible, a downlink mode is initiated instead.

Table 4.2: SSAEO rollout policy, $\pi_{\text{safe}}(s)$.

s_{tumbling}	$s_{\text{saturated}}$	$s_{\text{low power}}$	$s_{\text{buffer overflow}}$	Action
1	1	1	1	Charge
1	1	1	0	Charge
1	1	0	1	Desat
1	1	0	0	Desat
1	0	1	1	Charge
1	0	1	0	Charge
1	0	0	1	Downlink
1	0	0	0	Image
0	1	1	1	Desat
0	1	1	0	Desat
0	1	0	1	Desat
0	1	0	0	Desat
0	0	1	1	Charge
0	0	1	0	Charge
0	0	0	1	Downlink
0	0	0	0	Image

4.2.2.3 Backup Operator

After the rollout step in MCTS, a backup operator is applied to update the state-action value estimates of the nodes in the search tree. Vanilla MCTS utilizes an incremental averaging backup operator, where q is the return from the trajectory and $N(s,a)$ is the number of times the state-action pair has been visited:

$$\hat{Q}(s,a) = \hat{Q}(s,a) + \frac{q - \hat{Q}(s,a)}{N(s,a)} \tag{4.6}$$

This backup operator is most appropriate for MDPs with non-deterministic state transitions. While the models of the EOS scheduling problems utilized are complex, the state transitions are deterministic, so an incremental averaging backup operator may result in suboptimal state-action value estimates and require more simulations-per-step to converge to the optimal state-action value. Therefore, a maximization operator is also explored in this work, where the state-action value estimate is updated as follows:

$$\hat{Q}(s,a) = \max\{\hat{Q}(s,a), q\} \tag{4.7}$$

4.2.2.4 Training Data Construction

After MCTS finishes stepping through planning horizon, a trajectory of states and state-action value estimates, $\hat{Q}(s,:)$, at those states exists. Only the state-action value pairs associated with states that MCTS executes planning from are used for neural network regression. In other words, only the main search tree is used because states far removed from the main tree are visited very few times, resulting in poor state-action value estimates. In order for the trees generated by MCTS to be used, the intermediate state-action value pairs found using MCTS must be updated with the realized reward after MCTS finishes stepping through the EOS scheduling problem. Once an entire planning problem has been solved by MCTS, the reward received while stepping through the environment is used to compute new state-action values in the main tree for the actual actions selected. The intermediate state-action value pairs for each other action are left as they are. This

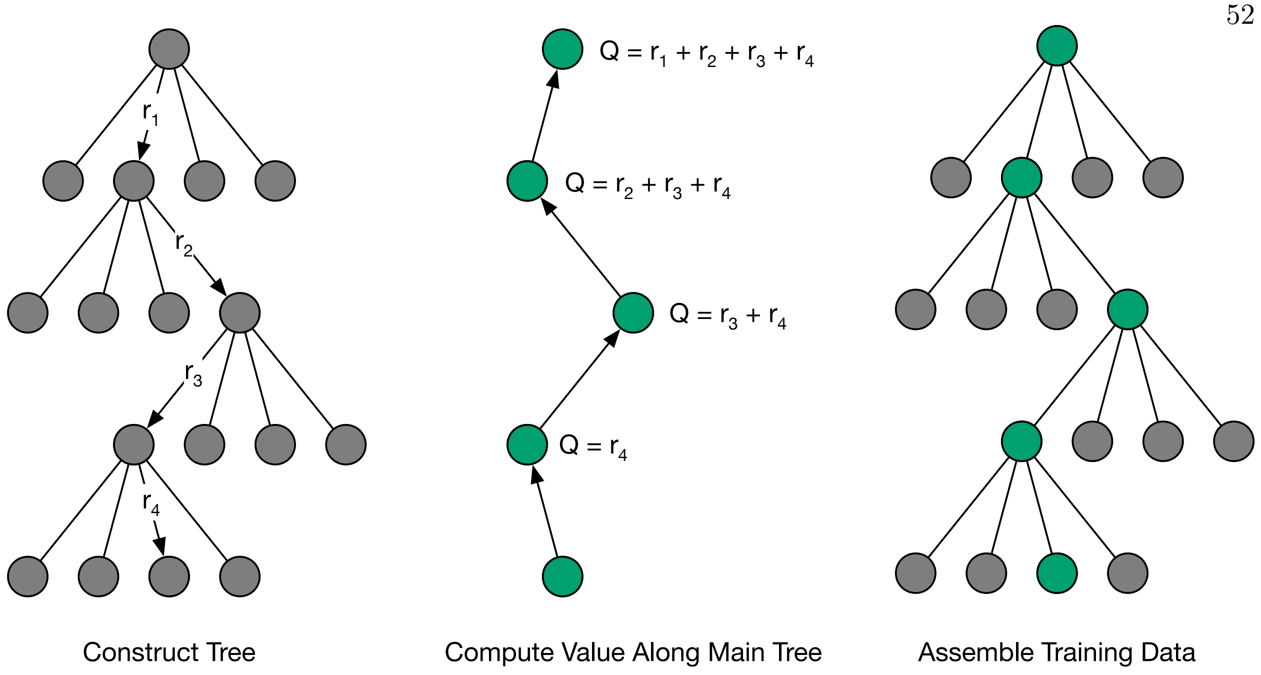


Figure 4.2: MCTS Final Value Computation

process is demonstrated in Figure 4.2, where the realized reward is backed up through the main search tree.

This is the final step performed before the value estimates generated by MCTS are added to the training data set. After all of the training data is generated, the training data is split up into a training set and a validation set. The training set contains 90% of the data and is used to train the neural network, while the validation set contains 10% of the data and is used to evaluate the performance of the neural network during training. The training data is shuffled before being split up into the training and validation sets to ensure the data is not ordered in any way.

4.2.3 Supervised Learning

After the training data is generated, supervised learning is applied over the training data set to generate a neural network approximation of the state-action value function, $Q_\theta(s, a)$. Hyper-parameters that relate to the activation function, width and depth of the network, learning rate, number of training epochs, etc. can be input into the algorithm to produce a number of neural networks. In this work, mean squared error is used for the loss function,

$$L(\theta) = \frac{1}{|\mathbf{Q}|} \sum_{s_i \in \mathbf{Q}} \Big(Q_\theta(s_i, a_i) - \hat{Q}(s_i, a_i) \Big)^2, \tag{4.8}$$

and the Adam optimizer is selected as the optimization algorithm to update the weights of the network(s). The neural networks are created and trained using the Keras[1] deep learning API in the Python programming language.

4.2.4 Validation and Benchmarking

After the neural networks are trained, they are validated in the environment using the policy in Equation (4.9). Each neural network is executed on the same set of N initial conditions and performance metrics are gathered. These performance metrics can be analyzed to determine the impact of the algorithm's hyperparameters as well as the parameters of the problem itself.

$$\pi(s) = \arg\max_a Q_\theta(s, a) \tag{4.9}$$

4.3 Solving the Simple EOS Scheduling Problem with MCTS-Train

4.3.1 Overview

The first problem solved using MCTS-Train is the simple EOS scheduling problem [69]. This section briefly describes the simple EOS scheduling problem in relation to the agile EOS scheduling problem and then discusses the use of MCTS-Train to develop a policy for the simple EOS scheduling problem. This includes hyperparameter searches for MCTS, hyperparameter searches for the supervised learning process, and various tests on the robustness of the trained policies.

In the simple EOS scheduling problem, a satellite in low-Earth orbit attempts to maximize the amount of science data collected and downlinked while avoiding resource constraint violations. In contrast to the agile EOS scheduling problem defined in Chapter 3, the simple EOS scheduling problem collects science data by pointing its instrument in the nadir direction. No imaging targets are collected in this problem. Data is collected throughout the entirety of the imaging mode.

[1] https://keras.io/

The state space of the simple EOS scheduling problem is the similar to that of the agile EOS scheduling problem, but does not include state information on the imaging targets:

$$\mathcal{S} = \mathcal{S}_{\text{sat}} \times \mathcal{S}_{\text{ground stations}}. \tag{4.10}$$

The representation of the state is also different:

$$\begin{split} s_i = (&{}^{\mathcal{E}}\mathbf{r},\ {}^{\mathcal{E}}\mathbf{v},\ \|\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}\|,\ \|{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}\|,\ \boldsymbol{\Omega},\ \text{battery},\ \text{eclipse},\ \text{buffer},\ \text{downlinked},\cdots \\ &\cdots\ \text{station 1 access},\cdots,\text{station 7 access},\ \text{planning horizon \% complete}) \end{split} \tag{4.11}$$

The position, velocity, attitude, attitude rate, and reaction wheel speed states are identical between the simple EOS and agile EOS scheduling problems. Furthermore, the battery, buffer, and downlinked states are also identical. However, instead of requiring that the network regress over the position and velocity directly to learn where the ground stations are, ground station access indicators are provided for each station. These ground station access indicators are represented as the percent of the previous planning interval that the ground stations are accessible. The planning horizon % complete state is also included to provide the neural network with information on how far along the planning horizon the agent is. This is important because the agent should be able to learn that it should downlink data when it is close to the end of the planning horizon.

The action space of the simple EOS scheduling problem is also slightly different

$$\mathcal{A} = \{\text{Charge},\ \text{Downlink},\ \text{Desaturate},\ \text{Image}\}. \tag{4.12}$$

The first three modes of the simple EOS scheduling problem are identical to those of the agile EOS scheduling problem. However, as previously stated, the imaging mode is a nadir-pointing mode where data collection does not require access to an imaging target as there are no imaging targets in this problem. The satellite points its instrument in the nadir direction and begins collecting data.

The reward function is formulated to reflect an EOS scheduling problem where the objective is to maximize the science data returned while managing resource constraints. The reward function is defined as the amount of data downlinked over each planning interval in megabytes, H_i, if no resource management failure occurs. Note that no reward is returned just for imaging, unlike the SSAEO scheduling problem. In the simple EOS scheduling problem, like the SSAEO scheduling problem, a failure constitutes a violation of resource constraints. The failure modes considered are zero charge in the battery, reaction wheels exceeding their maximum speeds, or an overflow in the data buffer. The expression for this is provided in Equation 3.8. If a failure does occur, a large penalty is returned.

$$R(s_i, a_i, s_{i+1}) = \begin{cases} H_i \text{ if !failure} \\ H_i + 1 \text{ if } t \geq t_{\text{max}} \text{ and !failure} \\ -1,000 \text{ if failure} \end{cases} \tag{4.13}$$

The spacecraft, ground station, and simulation parameters of the simple EOS scheduling problem are almost identical to those used for the SSAEO scheduling problem and are provided in Tables 3.3, 3.4, and 3.5. The only exceptions are a few of the spacecraft parameters, specifically those used for resource consumption. These are found in Table 4.3 and can be tied back to the difference in rollout policy parameters in Table 4.1. Because the maximum reaction wheel speed is almost twice as high in the simple EOS scheduling problem, the wheels can consume much more power. Therefore, more aggressive safety MDP limits are selected for management of the charge stored in the battery. As a result, the rollout policy executes trajectories where the battery is above 50% and the wheel speeds are below ∼3000 RPM. This allows MCTS to improve upon these power positive trajectories by trading off higher wheel speeds and lower battery charge for science collection and downlink. Also of note is the relationship between the instrument baud rate and the data buffer storage capacity. In the simple EOS scheduling problem, the decision-making agent can fill up its data buffer in 2,000 seconds of science data collection time. This is approximately 5.5 decision-making intervals. Therefore, a more aggressive constraint is utilized for the data buffer fill level.

A Basilisk Gymnasium interface for the simple EOS scheduling problem is also created. The simulation architecture is largely the same as the one presented for the SSAEO scheduling problem

Table 4.3: Simple EOS scheduling problem spacecraft parameters. Only the parameters that differ from the SSAEO scheduling problem (Table 3.3) are shown.

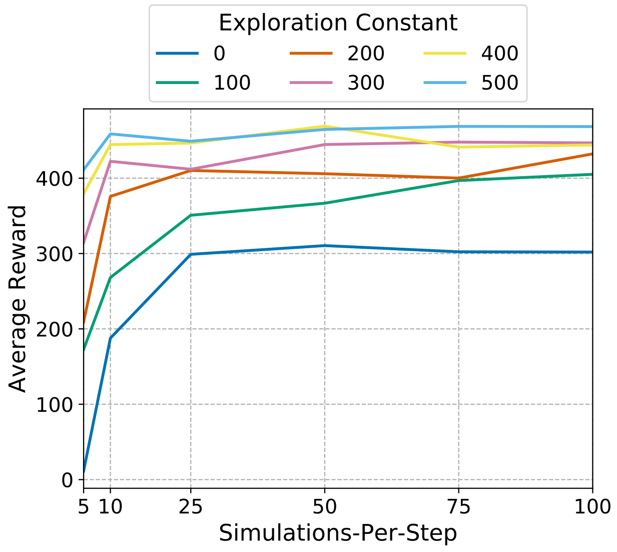
Attitude Control System	
Max Wheel Speeds	6000 RPM
Data & Communications System	
Data Buffer Storage Capacity	1 GB
Instrument Baud Rate	4 Mbps
Transmitter Baud Rate	4 Mbps

in Figure 3.6. However, there is no imaging target modeled or simple instrument controller.

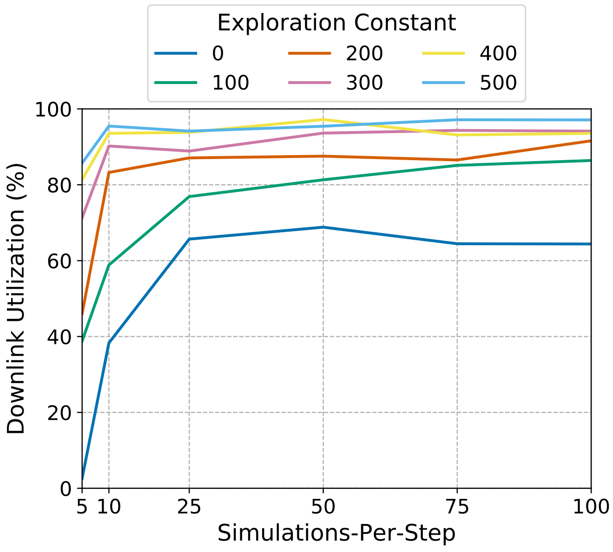
4.3.2 Monte Carlo Tree Search Hyperparameter Search

The first step of MCTS-Train is the generation of training data using MCTS. Before this can be done, though, a hyperparameter search of MCTS must be conducted to ensure high-performing policies are generated. In this section, a hyperparameter search is conducted to determine the best MCTS hyperparameter combination for generating training data. In the following search, different combinations of two key parameters for MCTS are tested: the exploration constant, ϵ, and the number of simulations-per-step. The exploration constant scales the exploration bonus during the search step, and the number of simulations-per-step determines how many simulations MCTS executes at each step through the environment. The hyperparameter search is also conducted for both types of rollout policies described in Section 4.2.2.2 - random and heuristic. Each hyperparameter combination is evaluated based on average episodic reward, downlink utilization, and the resource management success rate. The downlink utilization is a measure of how effectively the agent utilizes downlink opportunities. It is defined as the amount of time the agent downlinks data (i.e. sends data to a ground station, not spends in the downlink mode) divided by the amount of time downlink windows are available (i.e. the amount of time the spacecraft is within the ground station's field-of-view as specified by the elevation in Table 3.4).

Figure 4.3 displays both the average reward and average downlink utilization for MCTS with a heuristic rollout policy, $\pi_{\text{safe}}(s)$. To generate these plots, MCTS is executed on the same set of



(a) Average Episodic Reward

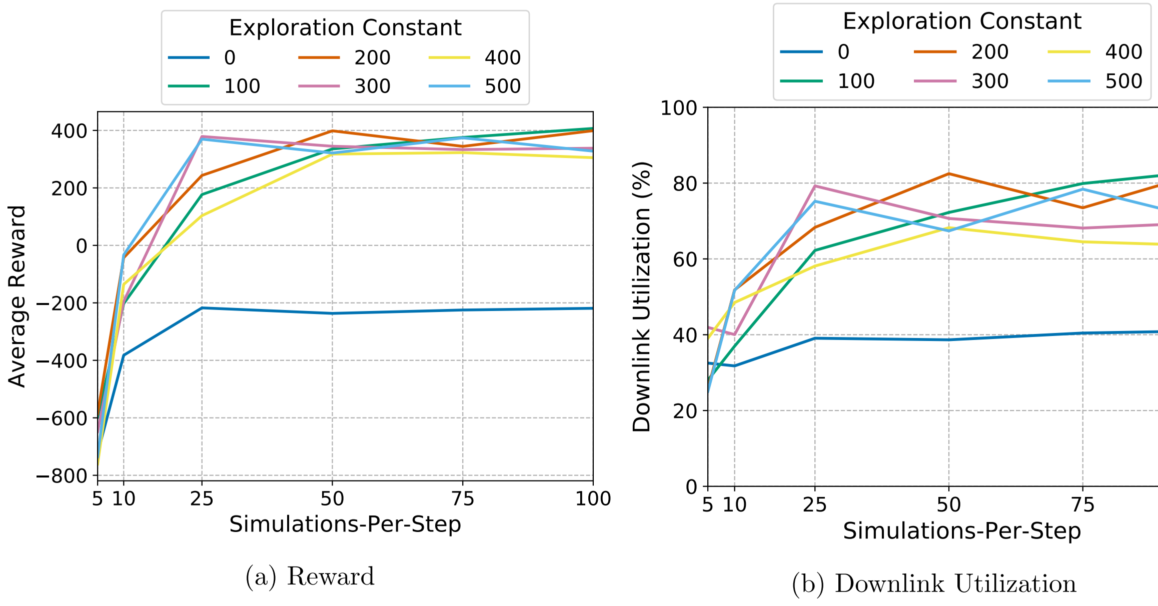


(b) Average Downlink Utilization

Figure 4.3: UCT Hyperparameter Search - Heuristic Rollout Policy

10 different initial conditions for each combination of ϵ and number of simulations-per-step. The results show that the average reward and downlink utilization are much more dependent on the exploration constant than the number of simulations-per-step. Adequate exploration ensures that the high-reward states discovered by the rollout policy are found again during the simulation step. Furthermore, adequate exploration allows MCTS to find higher value states than those discovered during rollout. While the exploration constant appears to be the most important hyperparameter, the number of simulations per step is important in terms of optimality. At 10 simulations-per-step, MCTS achieves a maximum average reward of 459 and downlink utilization of 95.5%. At 100 simulations-per-step, MCTS achieves a maximum average reward of 469 and downlink utilization of 97.1%. In the literature, MCTS is shown to converge to the optimal action as the number of simulations-per-step approaches infinity [57]. While MCTS achieves acceptable performance for this problem at 10 simulations-per-step, it takes at least an order of magnitude more simulations-per-step to converge to the optimal solution.

The same hyperparameter search is conducted for a random rollout policy, $\pi_{\text{rand}}(s)$, and the



(a) Reward

(b) Downlink Utilization

Figure 4.4: UCT Hyperparameter Search - Random Rollout Policy

results are provided in Figure 4.4. Both average reward and downlink utilization are much lower than the values generated by the heuristic rollout policy. In Figure 4.5, the resource management success rate is shown. Fifty simulations-per-step is the minimum required number for most exploration constants to achieve a 100% success rate. In the case of the heuristic rollout policy, the success rate is 100% regardless of the combination of hyperparameters. As hypothesized, the heuristic rollout policy does a far better job at avoiding resource constraint violations. As a result, MCTS only explores states where there is high reward as the resource constraint violations are avoided. Conversely, the random rollout policy provides no guarantees on avoiding resource constraints, so MCTS spends its simulation budget learning where the low reward states are.

4.3.3 State-Action Value Network Hyperparameter Search

As described in Section 4.2.3, state-action value training data is generated by MCTS. The selected MCTS hyperparameters for generating this data are an exploration constant of $\epsilon = 500$, 10 simulations-per-step, and a heuristic rollout policy. The selected hyperparameters balance the quality of the solutions with total execution time. For reference, MCTS can generate a solution

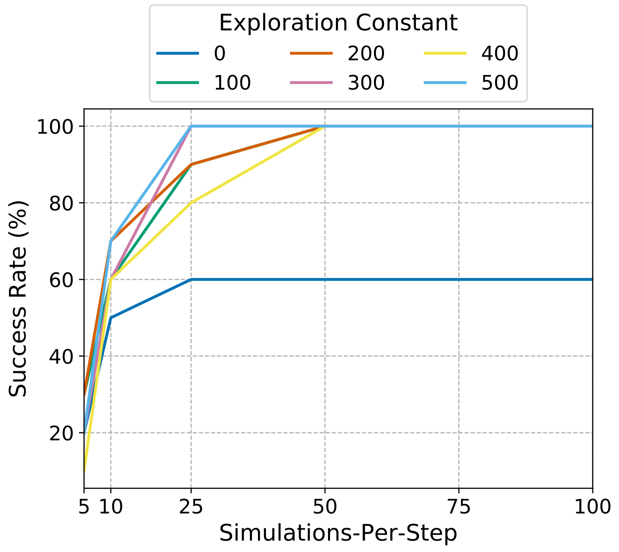


Figure 4.5: Resource Management Success Rate

for a single initial condition in 25 minutes using 10 simulations-per-step and a 2.8 GHz CPU. This can be extrapolated linearly for more simulations-per-step, demonstrating the need to keep this parameter as low as possible. The training data is generated using 1,200 unique initial conditions. To determine the best neural network architecture, a hyperparameter search is conducted over the number of hidden layers, the number of nodes per hidden layer, the activation function, dropout rate, and hyperparameters specific to the activation function. Each network is trained over 3000 epochs using a mean squared error (MSE) loss function.

The first hyperparameter search explores the performance of the state-action value networks when varying the number of hidden layers, the number of nodes per hidden layer, and activation functions of each layer. The dropout rate, which is the probability that a node will be dropped during a training epoch to avoid overfitting [70], is held constant at 0.25. Furthermore, the α parameter for Leaky ReLU is kept at the default of 0.3. The parameter α controls the slope of the Leaky ReLU activation function for $x < 0$. The performance is benchmarked using total reward, downlink utilization, and total time to execute. In Table 4.4, the performance of each hyperparameter combination is provided. For each combination, the top number is the average reward, and the bottom number is the average downlink utilization. Larger networks perform

Table 4.4: Leaky ReLU activation function general hyperparameter search. The top number is the average reward, and the bottom number is the average downlink utilization.

	Hidden Layers					
Nodes	1	2	3	4	5	6
100	309	356	383	409	365	416
	64.7%	75.6%	79.9%	84.4%	75.6%	87.4%
250	348	379	433	455	461	453
	72.5%	78.3%	89.3%	94.2%	95.9%	94.0%
500	344	402	450	460	459	462
	70.9%	82.4%	93.0%	95.5%	94.9%	95.8%

Table 4.5: Leaky ReLU activation function detailed search for 250 nodes per layer. The top number is the average reward, and the bottom number is the average downlink utilization.

Dropout		0.05				0.10				0.25			
	α	0.01	0.10	0.25	0.50	0.01	0.10	0.25	0.50	0.01	0.10	0.25	0.50
Hidden Layers	4	459	454	449	403	462	459	419	373	445	459	446	436
		94.9%	94.5%	93.0%	84.3%	96.0%	95.0%	87.0%	77.0%	92.7%	95.1%	92.6%	90.9%
	5	452	461	461	384	459	466	456	401	455	459	462	418
		94.3%	95.4%	95.5	79.4%	95.3%	96.7%	94.6%	84.4%	94.1%	95.4%	95.8%	86.6%
	6	455	462	463	403	455	462	453	362	451	463	460	453
		94.7%	95.9%	96.1%	84.4%	94.5%	95.8%	94.4%	76.3%	93.7%	95.8%	95.3%	94.4%

better on average. Between four and six hidden layers with 250 or 500 nodes each achieves the best performance, totaling between 2.0E5 and 1.3E6 trainable parameters. A smaller number of trainable parameters is preferred to increase the speed of training and execution.

A more detailed hyperparameter search is performed to determine the best combination of parameters when the number of nodes per hidden layer is held constant at 250 nodes per layer. The dropout rate, the number of hidden layers, and α are all varied during the hyperparameter search. As demonstrated in Table 4.5, the entire range of hyperparameters performs relatively well. Each dropout rate produces networks that achieve greater than 95% downlink utilization. Furthermore, each number of hidden layers produces networks that achieve greater than 95% downlink utilization. α is the one parameter that does not produce more than 95% downlink utilization for all values. In most cases, $\alpha = 0.50$ struggles to produce networks that can achieve greater than 90% downlink utilization.

In addition to performance, other metrics may give insight into the learned behavior of each neural network architecture. One such metric, the average amount of time each network architecture spends in each mode, can give insight into how well the networks have learned which planet-centered, planet-fixed position and velocity vectors are correlated with ground station access. In Figure 4.6, the average time (expressed as the percent time over each planning interval) several agents spend in each mode averaged over the 100 initial conditions is shown. For the purposes of readability, only 10 agents that demonstrate the breadth of solutions are selected. Furthermore, the selected agents are from Table 4.5 and achieve greater than 95% downlink utilization. Each agent spends about the same amount of time in the imaging mode. However, the time split between the charging, desaturation, and downlink modes varies widely for different architectures. Several architectures spend between 30-40% of the time in the downlink mode but achieve 95% downlink utilization. Other architectures spend 60-70% of the time in the downlink mode to achieve the same performance. Downlink windows are available for an average of 5.98% of the planning horizon, so in both cases the agents are spending more time attempting to downlink than is necessary. However, it is unlikely that the agents are randomly achieving this high reward considering the consistency of the performance over the 100 test conditions. This suggests that they have learned where the ground stations are located in terms of the planet-centered, planet-fixed position and velocity vectors to some degree. This is encouraging for future work, especially for multi-target scenarios in which the radius and velocity of multiple targets are input states and each target is its own action.

Another insight into the learned behavior of each network is demonstrated by the small variance in the percent of time each agent spends in the imaging mode. In Figure 4.6, each agent spends around 10-15% of the time in the imaging mode. This is due to the spacecraft filling up the data buffer quickly and only having a limited number of downlink opportunities available. As previously stated, the satellite can fill up its data buffer in only 5.5 decision-making intervals of continuous collection. The high-performing agents are limited by the size of the data buffer, and the other spacecraft modes do not include a penalty (other than power draw), so the spacecraft can split its time between the other three modes, converging to various local minima while achieving the same performance. This fact is the primary motivation in increasing the size of the data buffer relative to how much data a particular imaging mode can generate in the agile EOS scheduling

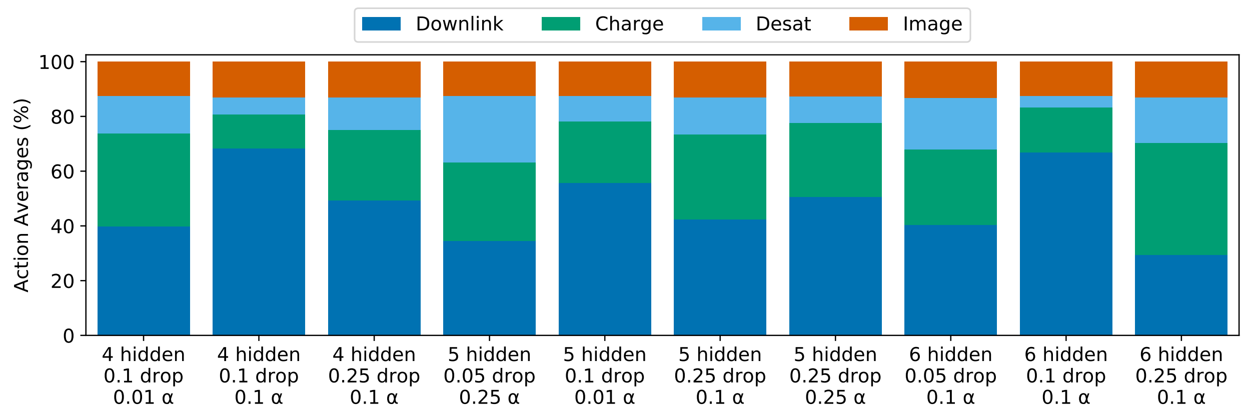


Figure 4.6: Average Action Percentages - Specific Search

problem. Another learned behavior demonstrated by a few networks highlights the dependence on the activation function. When the hyperparameter search in Table 4.4 is repeated for a hyperbolic tangent activation function, several architectures achieve an average reward of 1.00 and average downlink utilization of 0.00%. This is because the state-action value approximation converged to a local minimum where spacecraft charging was always the highest-value action in $Q_\theta(s, a)$.

4.3.4 Robustness

The data presented in the previous section makes a strong case for generalization within the training data distributions provided in Table 3.5 because each initial condition generated uses a different set of orbital parameters. However, the training data distributions do not cover all low-Earth orbits. Specifically, the semi-major axis is always initialized to 6,871 km. Furthermore, the initial epoch is held constant in training. Each planning horizon begins on May 4th, 2021. In this section, the effects of an erroneous orbit insertion into a higher semi-major axis orbit and a changing epoch are studied.

The performance of six neural network architectures is measured as the change in semi-major axis is increased from 0 to 2,000 km in increments of 100 km. In Figure 4.7, the average episodic reward increases with the semi-major axes of the orbits until Δa is between 500-750 km. This initially happens because the agents have more ground station access as the semi-major axis

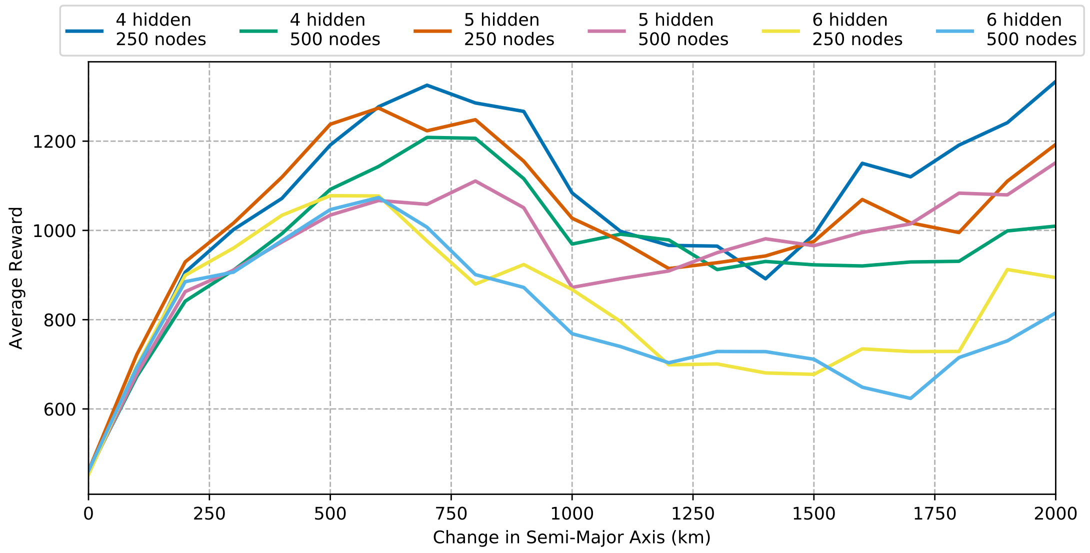


Figure 4.7: Average Episodic Reward as a Function of Semi-Major Axis

increases. Average episodic reward then begins to decrease as resource constraint failures begin to occur, as shown in Figure 4.8. The reward decreases before reaching a local minimum, where the reward penalty for failing to manage resources is offset by the increase in the length of the downlink windows. The resource management failures occur almost entirely due to data buffer overflows. While this may seem non-intuitive at first due to the increase in the length of the downlink windows, consider the fact that the frequency of new downlink windows decreases because of the larger semi-major axis. The agent anticipates upcoming downlink opportunities based on the planet-centered, planet-fixed position and velocity vectors. However, these do not occur and the agent overflows the data buffer. To rectify this issue, a state would need to be included in the state space that quantifies the relative size of the semi-major axis within the training distribution, normalized between $[0, 1]$. Regardless of the resource constraint violations for large deviations in semi-major axis, the trained neural networks can safely generalize up to a Δa of 500 km.

The initial epoch of each simulation is moved three, six, and nine months forward in time and the effect on performance, specifically resource management, is studied. When the epoch is moved to three and nine months in the future, some network architectures from Table 4.5 fail to manage

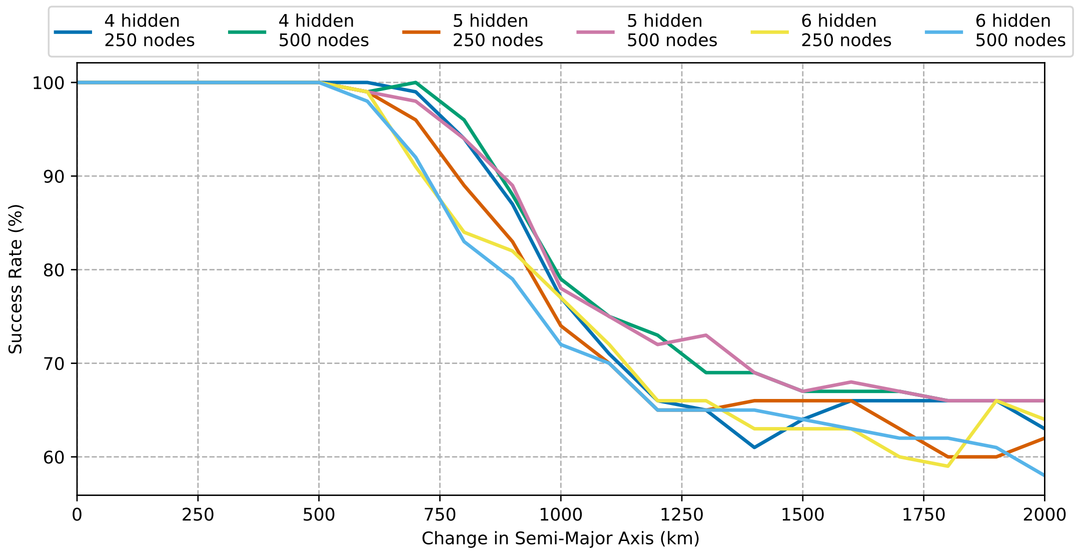


Figure 4.8: Resource Management Success Rate as a Function of Semi-Major Axis

power on-board the spacecraft and drain the battery, receiving a large reward penalty. For the 3-month change in epoch, 13 of the 36 trained agents produce resource management failures. For the 9-month change in epoch, 18 of the 36 trained agents produce resource management failures. When the epoch is moved six months into the future, the performance of the agents in Table 4.5 degrades the most. 29 of the 36 agents produce resource management failures. This is largely due to the change in the position of the sun relative to the Earth and the spacecraft and some agents becoming overfit on the relative position of the three during training. Furthermore, some agents anticipate ground station availability that never happens, suggesting they are overfit on the state that represents the percent of the planning horizon that has passed. This demonstrates the need to vary the epoch during training to prevent this type of overfitting. However, it is encouraging that a number of agents generalize to epochs outside the training distributions without producing failures.

Algorithm 8 Genetic algorithm.

```
1: Initialize population
2: Evaluate fitness of each individual in population
3: for generation 1 : $N$
4:     Generate offspring by mating and mutating population
5:     Evaluate offspring
6:     Add offspring to population
7:     Perform selection on population and offspring
```

4.3.5 Genetic Algorithm Comparison

To determine the optimality gap of Monte Carlo tree search and the resulting value networks for the given spacecraft configuration, a genetic algorithm is also tested on the same set of initial conditions. The genetic algorithm yields open-loop tasking solutions based on the expected environment, while MCTS and the neural networks yield closed-loop tasking solutions specific to observations generated by stepping through a real environment. Regardless, the genetic algorithm solution provides insight into how optimal the particular MCTS and neural network solutions are with respect to the reward function. The genetic algorithm is inspired by the biological processes of evolution and natural selection. The genetic algorithm begins by initializing a population of individuals, each of which is a sequence of actions for the EOS scheduling problem. Each individual is evaluated using a fitness function, which is simply the reward function of the corresponding EOS scheduling problem. The population of individuals then mates, and then their offspring are mutated. The offspring are then added to the overall population, and a selection operator is applied to select the best individuals within the population. This process repeats for a specified number of generations. The pseudocode for the genetic algorithm is provided in Algorithm 8.

The DEAP evolutionary computational framework is used to implement the genetic algorithm[2] . The DEAP framework has a number of pre-defined operators for selection, mating, and mutation. A one point crossover with a probability of 0.25 is used for mating. The selection operator utilized is the selection tournament in which the best individual from three individuals is returned. The population mates using a one point crossover operator, and the population mutates

[2] https://deap.readthedocs.io/en/master/index.html

Table 4.6: Genetic algorithm performance. The top number in each cell is the average reward and the bottom number is the average downlink utilization.

Population	Generations		
Size	45	100	200
10	445 94.0%	463 98.3%	447 96.2%
20	467 98.9%	472 99.8%	472 99.8%

using a uniform mutation operator where each sequence has a 0.25 probability of mutating, and each attribute of a mutating sequence has a 0.3 probability of mutating. The number of generations and population size are varied between 45-200 and 10-20, respectively. In Table 4.6, the genetic algorithm achieves the optimal solution of 472 reward and 99.8% downlink utilization. Only the first 10 of the 100 initial conditions used for the previous benchmarks are used due to computational limitations.

4.3.6 Final Comparison

Table 4.7 displays the reward, downlink utilization, execution time, and total number of Basilisk simulations required for each MCTS method implemented in this paper. The hyperparameter combination that achieves the highest reward is selected for each method. MCTS with a random rollout policy achieves its maximum reward at 100 simulations-per-step with a rather large optimality gap. MCTS with a heuristic rollout policy achieves its maximum reward at 75 simulations-per-step with an optimality gap of only 0.64%. After training, the neural networks achieve near-optimal performance with an optimality gap of 1.3%. However, the total execution time is several orders of magnitude less than any other algorithm implemented in this work. The value network is the only candidate algorithm in this work for on-board execution where execution speed is paramount on resource-constrained flight processors. Blacker et al. demonstrate that neural networks of a comparable size can execute on radiation hardened processors like the LEON3 in under 10 seconds [33].

Table 4.7: Comparison of Algorithms

	Random MCTS	Heuristic MCTS	Value Network
Avg. Reward	407	469	466
Avg. Downlink Util.	83.9%	97.1%	96.7%
Avg. Exec. Time	19,100 s	11,400 s	0.0672 s
Num. Simulations	4,500	3,375	0

The number of Basilisk simulations refers to the number of simulations required to generate one solution that achieves the demonstrated performance metrics in Table 4.7. For MCTS, this is computed by multiplying the number of simulations-per-step by the total number of planning intervals. Note that the state-action value networks achieve near-optimal performance by interacting with the environment only one time, and never with a simulated environment, selecting an action after each observation. MCTS and the genetic algorithm require many simulated environment interactions to solve the planning problem. Furthermore, the genetic algorithm yields open-loop tasking solutions. While MCTS technically yields closed-loop tasking solutions, it is given the truth model of the environment for the purposes of this work. Due to the power of neural networks to generalize across training data, the state-action value networks are able to interpolate and compute solutions to planning horizons with initial conditions they have never experienced before in a closed-loop implementation.

4.4 Solving the Agile EOS Scheduling Problem with MCTS-Train

4.4.1 Overview

The previous section provides an interesting case study for how MCTS-Train may be applied to a simple EOS scheduling problem. Several important conclusions may be drawn from it. First, a good rollout policy both improves performance and reduces the required computation. Artificial neural network approximations of the state-action value function that meet or exceed the performance of MCTS at multiple orders of magnitude less computation time are also possible. Insights are also shown regarding how the policies generalize. The trained neural networks generalize across the state space, and some trained networks perform just as well for semi-major axes and epochs outside the training distributions. Finally, both the MCTS and neural network solutions are near-optimal with respect to the reward function, which is determined through the use of the genetic algorithm (Algorithm 8).

However, several questions still remain, and several modifications should be made to the planning problem based on the insights gathered. The size of the data buffer relative to the amount of science data collected during a decision-making interval should be increased in order to formulate a problem where more precise management of the resources is required in order to maximize science data collected. Furthermore, the simple EOS scheduling problem is aptly named because it is a simplified version of the real problem, particularly in regard to its science objective. No imaging targets are considered. To address these issues, this section explores the single satellite agile EO scheduling problem. In the SSAEO scheduling problem, the objective of the decision-making agent is to maximize the number of imaged and downlinked targets while avoiding resource constraint violations. This problem is described in Chapter 3 and Reference [64]. The decision-making agent in the SSAEO scheduling problem has J imaging targets available in the set U for targeting at any given decision-making interval. A key question is to what value J should be set in order to maximize science return. If J is set to 1, then the agent only ever considers the next upcoming target. However, if J is set too large, then the action space becomes unnecessarily large as targets that are not available to the agent at the current decision-making interval are considered. Furthermore, the question of how the backup operator impacts performance also remains. This section investigates how MCTS may be used to tune J, and how different MCTS backup operators impact performance. The solutions generated by MCTS are again generalized with a neural network using the MCTS-Train pipeline. Finally, the genetic algorithm is again used to provide a performance benchmark for MCTS and the learned policies.

4.4.2 Action Space Parameterization

To parameterize the size of U, a constant target density of 45 possible targets per orbit (135 total targets in T over three orbits) is assumed. Only 45 decision-making intervals are utilized, so the agent will not be able to collect and downlink more than 45 targets. However, this target density provides a reward-rich environment in which the agent constantly must make trade-offs between high and low priority targets. It is worth mentioning that in addition to target density, the required number of targets in U is also a function of the length of the planning interval. A spacecraft that makes a decision every three minutes will be able to collect more targets than a spacecraft that makes a decision every six minutes, assuming the spacecraft can slew from target to target fast enough. This work only considers six-minute planning intervals.

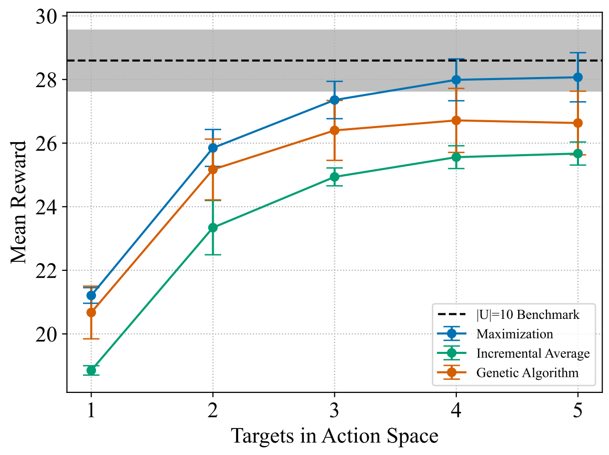
To determine how the size of U impacts performance, an experiment is conducted in which the number of targets in the action space is increased from one through five. A range of $|U| = \{1, \cdots, 5\}$ is selected because the spacecraft rarely has access to more than five targets. MCTS with an incremental average operator, MCTS with a maximization backup operator, and the genetic algorithm are applied to solve the problem for each size of U. For each $|U|$, the MCTS hyperparameters that balance performance and execution time are selected. For the incremental average backup, an exploration constant of $\epsilon = 10$ is selected. For the maximization backup, an exploration constant of $\epsilon = 20$ is used because of the higher state-action value estimates. The number of simulations-per-step for each $|U|$ is linearly increased from 15 to 35, with $|U| = 1$ at 15 simulations-per-step and $|U| = 5$ at 35 simulations-per-step. These numbers are roughly based on the hyperparameter searches in the previous section, and the increase in simulations-per-step is selected due to the increase in problem complexity due to additional actions in the action space. The number of times future state-action pairs are visited is on the order of

$$\frac{1}{|\mathcal{A}|^{\text{depth}}}, \tag{4.14}$$

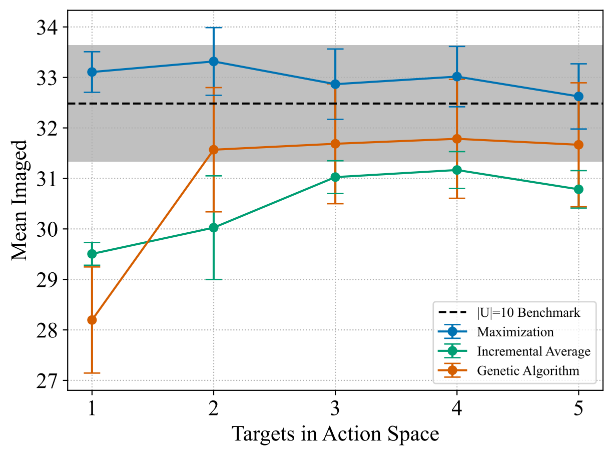
where the depth is the depth of the search. To equalize exploration between the different experiments, the number of simulations-per-step is marginally increased to account for this decay. The simple EOS MCTS hyperparameter searches show that additional simulations-per-step in the EOS scheduling problem do not result in a large difference in the quality of MCTS solutions. Therefore, the smallest number of simulations-per-step is selected for each $|U|$ such that an MCTS performance plateau is achieved.

Once again the genetic algorithm is implemented for comparison purposes. The genetic algorithm is initialized with a population of 80 for $|U| = 1$ and is linearly increased to 160 for $|U| = 5$ to account for the increase in problem complexity as the action space grows. For each size of U, the number of generations is set to 100. Note that the initial population size is much larger than that of the simple EOS scheduling problem (Table 4.6). This is done because the SSAEO scheduling problem is more complex than the simple EOS scheduling problem, with far more science modes possible due to the change in data buffer size and instrument baud rate.

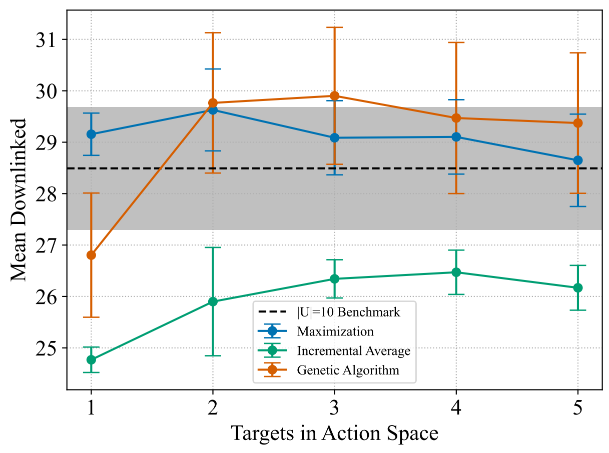
In Figure 4.9, the average reward, number of imaged targets, and number of downlinked targets for each size of U is plotted, along with the 95% confidence intervals for each. These metrics are provided to give detailed insights into the performance of the decision-making agents. The maximum possible number of imaged targets is 45. However, this is impossible to obtain because of the data buffer constraint. Therefore, MCTS with the maximization backup operator is used to generate a value for each metric's maximum expected value (75 simulations-per-step, 10 targets in the action space). This is provided using the dotted black line, with the 95% confidence intervals provided in gray. The solid blue line is generated from MCTS using the maximization backup operator. The solid green line is generated from MCTS using the incremental averaging backup strategy. Finally, the orange line is generated using the genetic algorithm. The reward asymptotically approaches the maximum possible reward for each backup strategy as $|U|$ increases. However, the maximization backup strategy consistently achieves more reward. The maximization backup operator also images and downlinks more targets on average. This is because the incremental averaging operator averages in the return from future low-value states. The maximization backup operator does not average in this return and instead sets the action-value estimate to the maximum return found during search. Consider a scenario in which the agent is close to filling



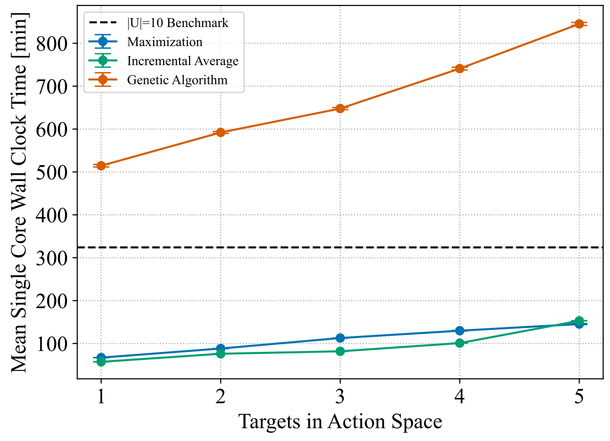
(a) Mean reward.



(b) Mean number of imaged targets.



(c) Mean number of downlinked targets.



(d) Mean single core wall clock time.

Figure 4.9: MCTS performance metrics for different sizes of U with associated 95% confidence intervals.

up the spacecraft's data buffer. The state-action value estimates along the optimal trajectory of actions will be lower than the true optimal state-action value the closer the agent is to filling up the data buffer. In contrast, the maximization operator will not consider the low return from a data buffer overflow until it reaches a state in which it will overfill the data buffer at the next time step if it takes an image.

An interesting result in Figure 4.9 is that the mean number of imaged and downlinked targets are relatively constant for each backup operator as the size of U increases. The average reward, however, increases as $|U|$ increases. The reason for this disparity is that as more targets are included in the action space, MCTS has more of an opportunity to select high priority targets over low priority targets. The target priorities are uniformly sampled from a range of one to three. With one target in the action space, there is a probability of 1/3 that the target will be priority one. However, this probability increases with the size of U using the following equation: $1-(2/3)^{|U|}$. With five targets in the action space, the probability that MCTS will have at minimum one priority one target is approximately 0.868.

Another observation to note is that the maximum number of imaged targets hovers around 33. Therefore, it is possible for the decision-making agents to select the imaging mode roughly 75% of the time. This is a major increase in contrast to the simple EOS scheduling problem where the imaging mode is selected for at most 15% of the time due to the small size of the data buffer. The SSAEO scheduling problem therefore requires more precise management of power, reaction wheel speed, and data buffer storage and is a more appropriate problem for future chapters.

In terms of reward, the genetic algorithm performs better than MCTS with the incremental averaging operator, but not as good as MCTS with the maximization operator. Generally speaking, this is true for the number of imaged targets as well. However, the genetic algorithm usually downlinks more targets than the maximization operator and incremental averaging operator. The GA seems to take better advantage of downlink opportunities, but does not leverage the imaging target priorities quite as well. The mean single core wall clock time of each algorithm is plotted in Figure 4.9d. The MCTS computation time is typically between 10-20% of that of the genetic algorithm. The reason for this is that MCTS leverages expert knowledge in the rollout policy to find high-value states, so the MCTS algorithm is more sample efficient. This implementation of the genetic algorithm does not have an analogous mechanism. It should be noted that the computation times are on the order of hours long for each algorithm. If a faster simulator were used, especially one that makes linear assumptions about the dynamics of the problem, these simulation times could be greatly reduced.

In Figure 4.10, the normalized frequency of the number of accessible targets at any given

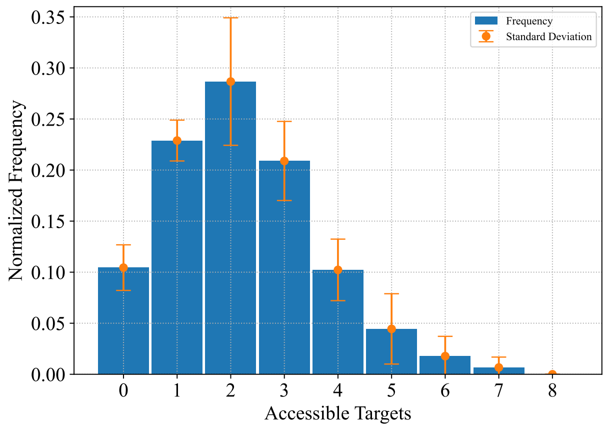


Figure 4.10: Normalized frequency of the number of accessible targets at each planning interval. $|T| = 135$.

step is provided. The 1σ standard deviation of the frequencies between different initial conditions is plotted in orange. This is computed by generating the normalized frequencies for each unique initial condition and computing the associated standard deviations between each initial condition. The agent has no targets available to image in the next decision-making interval approximately 10% of the time. The agent has two targets available to image in the next decision-making interval about 28% of the time. Finally, the agent rarely has eight targets available for imaging and never has nine targets available for imaging. Therefore, ten targets in the action space provides MCTS with the maximum amount of targets required to get the maximum reward at any step, with some margin added. The frequencies of available targets also shed light on the curves in Figure 4.9. The spacecraft has five or more targets available for imaging less than 10% of the time. The magnitudes of the frequencies between one through four targets is much larger and therefore has a much higher impact on the performance.

4.4.3 State-Action Value Function Approximation

Similar to the simple EOS scheduling problem, a hyperparameter search is conducted over the number of nodes, number of hidden layers, activation function, dropout rate, and α parameter (specific to the Leaky ReLU activation function) to determine the appropriate artificial neural

Table 4.8: Neural network hyperparameters for the SSAEO scheduling problem.

Parameter	**Value**
Nodes Per Hidden Layer	{50, 100, 200, 300}
Hidden Layers	$\{1, \cdots, 5\}$
Activation Function	{Leaky ReLU, tanh}
α	{0.1, 0.2}
Dropout	{0.01, 0.1}
Epochs	10,000
Batch Size	45,000
Loss Function	Mean Squared Error

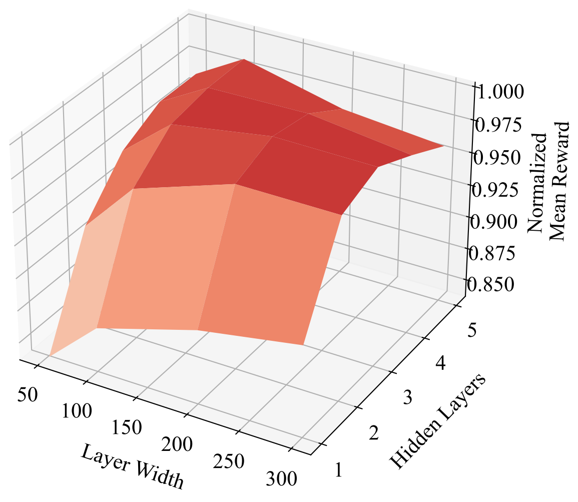
network hyperparameters for approximating the state-action value estimates generated by MCTS. The complete network parameters are included in Table 4.8. The training data generation and neural network hyperparameter search were conducted on a computer with a Windows operating system, a 24-core AMD Threadripper 3960x processor, NVIDIA RTX 3090 graphics card, and 128 GB of RAM. In general, training data generation takes between 1-2 days with multiprocessing being utilized. Each network requires approximately 5-10 minutes of training using the GPU. If the CPU is utilized instead, this training time increases dramatically.

To approximate the state-action value estimates, fully connected feedforward neural networks with linear output layers are implemented with various hyperparameters. Dropout is again added to each hidden layer to avoid overfitting [70]. Two dropout rates are explored - 0.01 and 0.1. If the dropout rate is too low, it will not have the desired effect. If the dropout rate is too large, the network may not sufficiently learn. These dropout rates are significantly smaller than the simple EOS scheduling problem. Larger dropout rates appeared to hurt performance more for the SSAEO scheduling problem than the simple EOS scheduling problem. In order to determine the appropriate number of hidden layers and width of these layers, a hyperparameter search is conducted over a range of these parameters. The number of hidden layers is increased from one to five. Four network widths are considered - 50, 100, 200, and 300 nodes. These ranges of values were iteratively increased until a depreciation in performance was present. In general, it is desirable to keep the network as small as possible to increase the speed of training and decrease the memory footprint and execution time of the network, especially when on-board execution is desired.

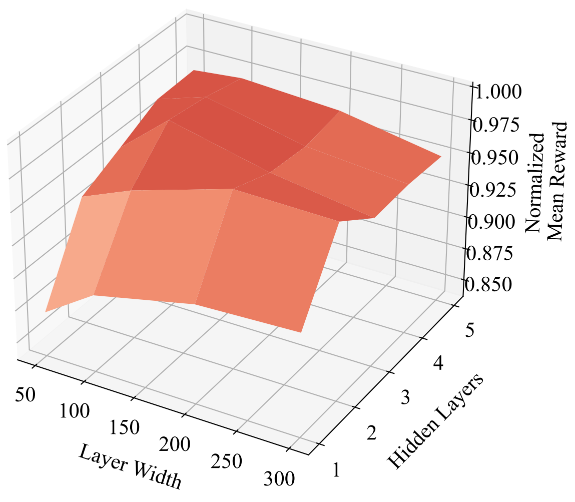
Both the Leaky ReLU and hyperbolic tangent activation functions are considered. The hyperbolic tangent activation function is a popular activation function that may be used for regression, but suffers from the vanishing gradient problem for deep networks [71]. Therefore, Leaky ReLU is also considered, which has become a popular choice for regression because it does not suffer from the vanishing gradient problem (for sufficiently large values of α) or the dying ReLU problem associated with the ReLU activation function. The α-parameter in the Leaky ReLU activation function is the slope of the activation function for $x < 0$. Two α-parameters are explored - 0.1 and 0.2. If the α-parameter is too small, Leaky ReLU may also suffer from the vanishing gradient problem.

MCTS is used to generate 45,000 data points (1,000 unique planning horizons solved). 90% of the data is used for training, and 10% of the data is used for validation. The data generated by MCTS is not pre- or post-processed (outside of MCTS-Train updating the state-action value estimates after it steps through the environment). The networks are trained for 10,000 epochs using the mean squared error (MSE) loss function, a popular choice of loss function for regression problems. In general, the mean squared error for the training and validation set is between 10-20. The mean absolute error is also logged and is found to be in between 2-4. Overfitting is observed as the networks became larger, but in general this is not an issue for smaller networks with 1-3 hidden layers and 50-200 nodes.

To validate the performance of each trained neural network, the state-action value networks are used to generate the policy in Equation 4.9, which is executed on a standard set of initial conditions. An example of this hyperparameter search is shown with the number of targets in the action space set to three, $|U| = 3$. The same hyperparameter search is performed for the other sizes of U, but these are not shown here for brevity. The general trends shown here are consistent among the other numbers of targets in the action space. In Figure 4.11, a surface plot is shown that plots normalized reward against the number of nodes per hidden layers and number of hidden layers. The normalized reward is the average reward divided by the reward of the best hyperparameter combination. A surface plot is generated for each activation function. However, for each data point



(a) Leaky ReLU activation function.



(b) Hyperbolic tangent activation function.

Figure 4.11: Hyperparameter surface plots for mean reward normalized by the mean reward of best performing network. $|U| = 3$.

in the surface plot, the reward is averaged among the remaining hyperparameters. In the case of Leaky ReLU, the α parameter and dropout rate rewards are averaged. In the case of hyperbolic tangent, only the dropout rate reward is averaged. In general, performance is not as sensitive to the dropout rates and α parameters as it is to the width of the hidden layers and number of hidden layers.

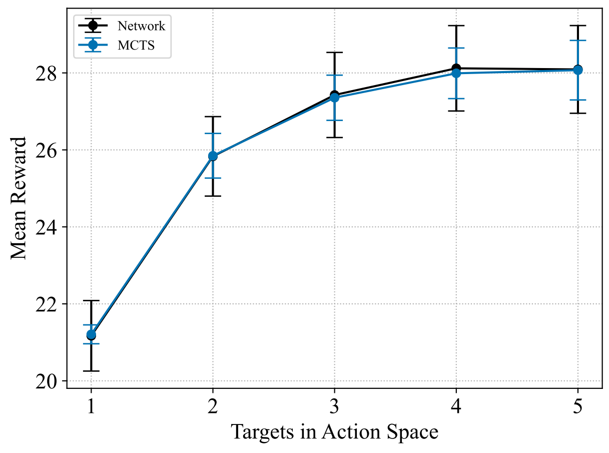
A general trend can be extracted from the plots in Figure 4.11. One to two hidden layers is typically insufficient to produce high-performing policies, particularly when small layer widths are used. Furthermore, performance is relatively poor for large layer widths and large numbers of hidden layers, albeit not as poor as the former case. In the former case, the network does not have enough parameters to sufficiently approximate the state-action value estimates. In the latter case, the network has too many parameters and becomes overfit. In the region that excludes these two conditions, performance is relatively high. It's also worth mentioning that no combination of layer widths or hidden layers produces less than 85% normalized reward. Therefore, the performance is relatively robust to the network architecture, which is an encouraging result.

4.4.4 Comparison Between MCTS and Learned Policies

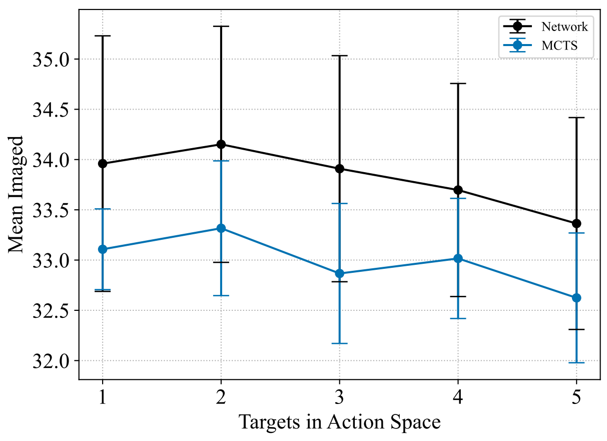
Once the state-action value network hyperparameterization is performed for each $|U|$ as described in the previous section, the best state-action value networks for each $|U|$ are selected to compare against MCTS. Each of these networks are trained using data generated from MCTS with the maximization operator solving approximately 1,000 unique initial conditions. The average reward, number of imaged targets, number of downlinked targets, and the wall clock time, along with the 95% confidence intervals, for both MCTS and the best-performing neural networks are plotted in Figure 4.12. For each $|U|$, the neural network policies image and downlink anywhere between 0.5 - 1.0 more targets on average. There is a significant overlap in the confidence intervals, but the network policies image and downlink more targets for every number of targets in the action space. Because of this and the fact that the reward achieved by MCTS and the neural network policies is almost identical, it is likely that the neural network policies are slightly worse at imaging the high-priority targets. However, this difference is quite small, so it can be concluded that the neural network policies approximate the MCTS policy very well. Finally, the learned policies find a solution three orders of magnitude faster than MCTS.

4.4.5 Robustness to New Target Densities

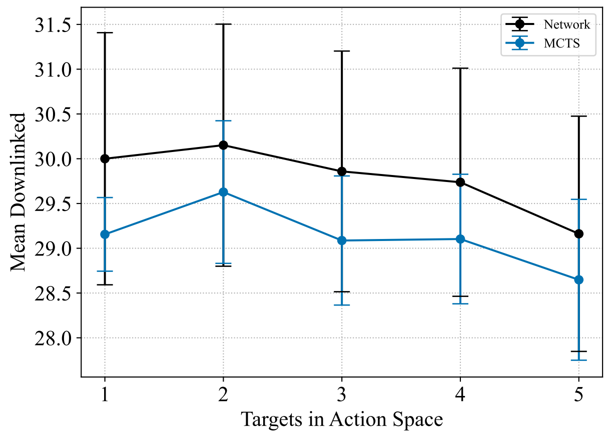
In the last experiment, the robustness of the trained state-action value function approximators is explored for various sizes of the total target set, T. During training, $|T|$ is set to 135. However, in a real operational scenario, this number will not be constant. The trained networks are deployed in environments with $|T| = \{45, \cdots, 270\}$ in increments of 45. The results of this experiment are presented in Figure 4.13a where the average reward and associated 95% confidence intervals are plotted. At $|U| = 1$, there is little difference between the performance of the trained network for the different numbers of targets in the total target set. This is because the agent is only ever considering the next upcoming target. Adding or subtracting targets typically does not impact the performance, except when $|T| = 45$. In this case, the targets are very sparse and performance is limited by the



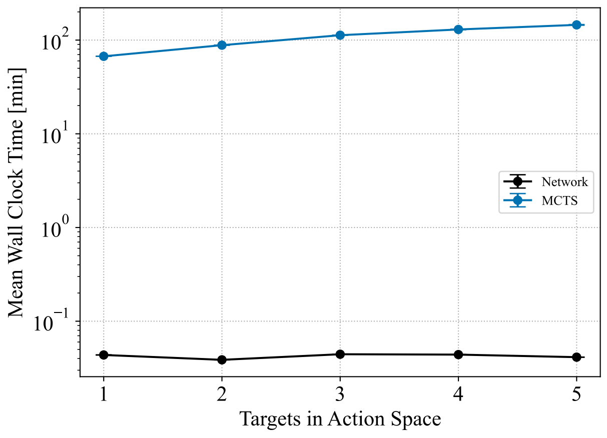
(a) Mean reward.



(b) Mean number of imaged targets.



(c) Mean number of downlinked targets.



(d) Mean wall clock time.

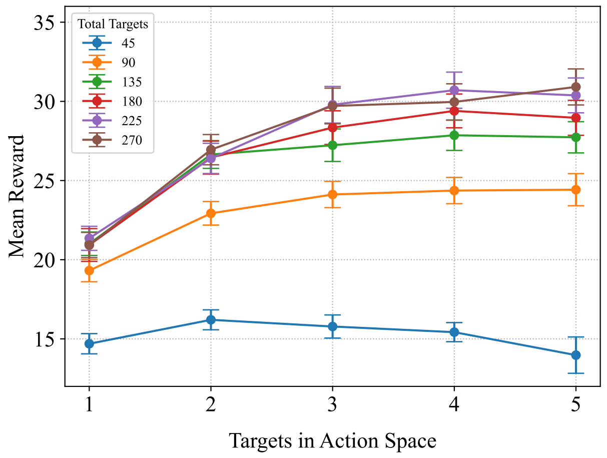
Figure 4.12: MCTS and learned policy performance for different sizes of $|U|$.

number of targets available for imaging and downlink, not the agent's ability to discern between different priorities. However, as the number of targets in the action space increases, a separation in performance emerges for $|T| = \{90, 135, 180\}$. These three target densities are fairly distinct from one another, although there is some overlap in confidence intervals. For $|T| = \{225, 270\}$, there is a lot of overlap in the means and confidence intervals for all numbers of targets in the action space. This is expected because priority one targets become more available as the target density increases. In Figure 4.14, the normalized frequencies of available targets at each planning interval are plotted for $|T| = 270$. In contrast to Figure 4.10, the most commonly available number of targets is four as opposed to two. The agent has higher priority targets available to it more frequently, which increases the average reward.

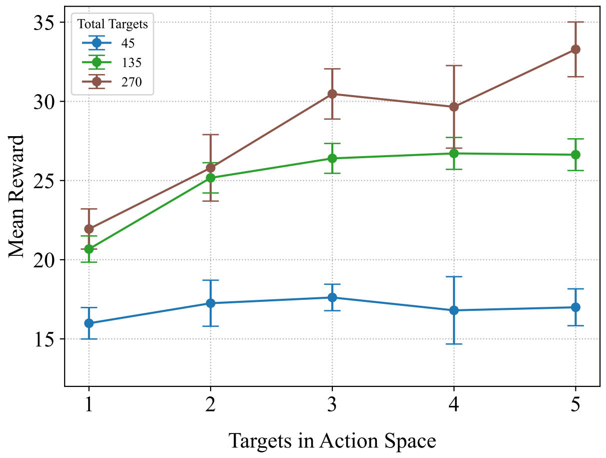
A comparison to the genetic algorithm is performed once more, this time on different target densities. In Figure 4.13b, the average reward of the genetic algorithm is plotted for $|T| = \{45, 135, 270\}$, which provides a lower, middle, and upper bound on the performance of the GA. At $|T| = 45$, the performance of the trained networks and the GA immediately plateau because the MDP limits the ability of either decision-making agent to prioritize high-priority targets over low-priority targets. However, the GA performs slightly better at this target density because it makes no assumptions about the underlying density. At the nominal size of T, $|T| = 135$, the GA fails to match the reward of the networks trained using MCTS, which is discussed in the original comparison. However, at $|T| = 270$, the GA begins to slightly outperform the trained networks again. The networks are trained for a nominal density of 135 total targets. Therefore, the state-action value outputs assume a given target density, which may result in sub-optimal results. The GA makes no such assumptions and simply searches for the sequence of actions that will maximize the reward signal, albeit at a high computational cost. Hypothetically, the networks could be retrained using MCTS assuming the lower or higher target density, and the discrepancy in performance would disappear while maintaining the low-computational cost of the trained networks.

4.5 Conclusion

This chapter describes the MCTS-Train pipeline and presents two case studies on using the pipeline to parameterize the EOS scheduling environments and generate artificial neural network approximations of the state-action value function generated by MCTS. The trained neural networks are able to meet or exceed the performance of MCTS for a fraction of the computational cost after training, generalize across the training distribution, and even generalize to distributions outside those used in training. MCTS and the trained agents are also shown to be comparable to a genetic algorithm. It is for these reasons that the MCTS-Train pipeline is a promising approach for on-board planning and scheduling of the EOS scheduling problem.



(a) Network trained on nominal target density.



(b) Genetic algorithm.

Figure 4.13: Average reward for different sizes of T

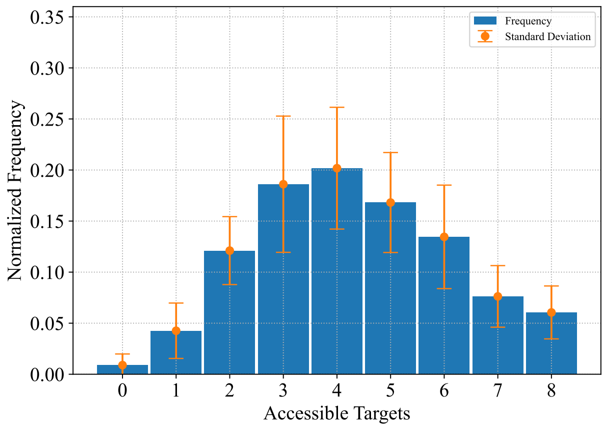


Figure 4.14: Normalized frequency of the number of accessible targets at each planning interval. $|T| = 270$.

Chapter 5

A Comparative Analysis of Reinforcement Learning Algorithms for EOS Scheduling

5.1 Introduction

The previous chapter introduces MCTS-Train and provides two case studies on how the algorithm may be deployed on EOS scheduling problems to parameterize the environments and produce near-optimal policies that may be executed on-board spacecraft in milliseconds. Furthermore, the previous chapter investigates how these networks perform outside the nominal training distributions and compare to solutions computed by a genetic algorithm. However, several questions remain unanswered. First and foremost, it is unclear how MCTS-Train compares to other state-of-the-art reinforcement learning algorithms in regard to performance, performance variance, training wall clock time, and safety. In fact, this is a common issue facing the literature today. Each paper studying reinforcement learning for EOS scheduling implements a different algorithm on a different planning problem. Examples of EOS scheduling work using different algorithms include asynchronous advantage actor-critic (A3C) [49], REINFORCE [50], deep Q-networks (DQN) [51], PPO [52], and Monte Carlo tree search [54]. While each of the aforementioned works provide novel contributions to the field of EOS scheduling, any comparisons are relatively limited in scope. Harris and Schaub compare PPO and shielded PPO (SPPO), demonstrating that SPPO improves resource constraint satisfaction and convergence time [52]. Eddy and Kochenderfer provide an excellent comparison for MCTS, forward search, rule-based, and graph-based methods, but do not consider deep reinforcement learning algorithms. In addition to these gaps in the literature, the relationship between the different DRL algorithms, their hyperparameters, and performance variance between different initial seeds is not well documented for EOS scheduling problems.

In addition to the shortcomings in regard to DRL comparisons for EOS scheduling in the literature, the impact of the length of the planning horizons (i.e. number of decision-making intervals) and the parameters of the reward function are not explored either. The majority of authors train their decision-making agents on relatively small planning horizons (i.e. a few orbits) and do not investigate how these agents perform long-term (i.e. days of operations). Furthermore, little to no experiments are performed to tune the components of the reward functions. For a given state and action space, different reward functions can produce wildly different behaviors in decision-making agents, depending on the problem.

To address these shortcomings, this chapter provides a comprehensive comparison, complete with hyperparameter searches, between various reinforcement learning algorithms for two EOS scheduling problems: the aforementioned SSAEO scheduling problem and a new multi-sensor EOS scheduling problem. To evaluate the impact of the length of the decision-making interval, the multi-sensor EOS scheduling problem is evaluated for two numbers of decision-making intervals, $|I| = 45$ and $|I| = 90$. The SSAEO scheduling problem is only evaluated for $|I| = 45$. This work implements PPO, SPPO, advantage actor-critic (A2C), MCTS-Train, and DQN to solve each scheduling problem. Hyperparameter searches are shown for each algorithm, and each algorithm is compared on the basis of performance across various hyperparameters, performance variance between training seeds, and wall clock time. Each algorithm is also compared to a genetic algorithm performance baseline. Finally, PPO and MCTS-Train are used to evaluate long-term deployment and training of the decision-making agents, as well as the structure of the reward function. Part of this work is published in Reference [72].

5.2 Problem Formulations

5.2.1 Single Satellite Agile EO Scheduling Problem

The SSAEO scheduling problem is described in detail in Chapter 3. However, several modifications are made to the environment in this chapter. First and foremost, the reward function $R(s_i, a_i, s_{i+1})$ is modified to have a minimum reward sum of -1 and a maximum reward sum of 1 as follows:

$$R(s_i, a_i, s_{i+1}) = \begin{cases} -F & \text{if failure} \\ \frac{A}{|I|} \sum_j^{|\boldsymbol{T}|} H(d_j) & \text{if } \neg\text{failure} \wedge a_i \text{ is downlink} \\ \frac{B}{|I|} H(w_j) & \text{if } \neg\text{failure} \wedge a_i \text{ is image } c_j \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The failure constant, F, is set to 1. A downlink constant, A, is utilized to weight the positive reward component from downlink. An imaging constant, B, is utilized to weight the positive reward component from imaging. $A = 0.9$ and $B = 0.1$ is selected to provide more weight towards downlinking images as opposed to only collecting images on-board. Furthermore, each component is normalized by the number of decision-making intervals, $|I| = 45$, because the number of decision-making intervals is the limiting factor regarding the reward (i.e. there are more targets in the set T than there are decision-making intervals, $|T| > |I|$). Another change to the environment parameterization is a modification to the range of ascending nodes in Table 3.5. The range of ascending nodes is modified to be $[0, 360)$ deg as opposed to $[0, 20]$ deg. This is done to increase the generalizability of the trained networks. Finally, the set of imaging targets in the action space, U, is sized to $|U| = 3$. Chapter 4 demonstrates that the performance gains for $|U| > 3$ are minimal.

5.2.2 Multi-Sensor EOS Scheduling Problem

A multi-sensor EOS scheduling problem is also formulated to provide a basic training environment in comparison to the SSAEO scheduling problem that uses a target imaging objective as opposed to a quasi-area imaging objective. In the multi-sensor EOS scheduling problem, a nadir-pointing satellite attempts to maximize the sum of imaging targets collected with one of two sensor types, A or B, while managing power and reaction wheel speeds. No data buffer constraints or data downlink is considered, which makes the reward less sparse. Like the SSAEO scheduling problem, the set of all imaging targets is referred to as T. However, the satellite only ever considers the next upcoming target. The planning horizon is divided into two separate lengths: 45 and 90 decision-making intervals. Each decision-making interval lasts for three minutes, which is different from the six-minute intervals explored thus far.

5.2.2.1 State Space

The state space for the multi-sensor EOS scheduling problem is defined as:

$$\mathcal{S} = \mathcal{S}_{\text{sat}} \times \mathcal{S}_{\text{targets}}, \tag{5.2}$$

where state space of the satellite is defined as:

$$\mathcal{S}_{\text{sat}} = \mathcal{S}_{\text{pos}} \times \mathcal{S}_{\text{vel}} \times \mathcal{S}_{\text{att}} \times \mathcal{S}_{\text{wheels}} \times \mathcal{S}_{\text{power}} \times \mathcal{S}_{\text{eclipse}}. \tag{5.3}$$

The state returned to the decision-making agent at decision interval i is defined as $s_i \in S$:

$$s_i = ({}^{\text{SEZ}}\mathbf{r},\ {}^{\text{SEZ}}\mathbf{v},\ \|\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}\|,\ \|{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}\|,\ \|\boldsymbol{\Omega}\|,\ \text{battery, access, sensor, eclipse}). \tag{5.4}$$

The position and velocity, ${}^{\text{SEZ}}\mathbf{r}$ and ${}^{\text{SEZ}}\mathbf{v}$, of the satellite are expressed in the topocentric horizon coordinate system, or SEZ frame. This coordinate system is defined relative to the imaging target and is selected to ensure the policy is target agnostic. Attitude information is provided in the form of spacecraft attitude, angular velocity, and reaction wheel speeds. The attitude of the satellite is provided as the magnitude of the modified Rodriguez parameters (MRPs) [73] $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{R}}$, which is the rotation from the reference frame to the body frame. The magnitude of the angular velocity of the satellite, ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$, is provided in the body frame. The magnitude of the reaction wheel speeds is given by $\|\boldsymbol{\Omega}\|$. Several states are also included for the purposes of power management. The percent charge of the battery is provided using the battery variable, and the eclipse variable is a binary variable that indicates whether the satellite is in eclipse. The last few states deal with target access and which type of sensor the target should be imaged with. The access variable is a binary variable that indicates whether the satellite has access to the next upcoming imaging target, and the sensor variable is a binary variable that indicates which type of sensor the satellite should image the upcoming target with.

5.2.2.2 Action Space

Like the state space, the action space is designed to fulfill the science objectives and manage the resource constraints. Each action represents a distinct spacecraft mode. Descriptions of each are provided below:

(1) **Charge**

- The satellite turns off its imager and points its solar panels at the sun to charge the battery.

(2) **Desaturate**

- The satellite turns off its imager and points its solar panels at the sun. Reaction wheel momentum is mapped to thrust commands, which are executed to remove momentum from the wheels.

(3) **Image with sensor A**

- The satellite turns on imager A, and points it in the nadir direction. An image is taken when requirements are met.

(4) **Image with sensor B**

- The satellite turns on imager B, and points it in the nadir direction. An image is taken when requirements are met.

5.2.2.3 Reward Function

A piecewise reward function is developed to mathematically formalize the objectives of the multi-sensor EOS scheduling problem. Like the SSAEO scheduling problem in this chapter, the reward function is formulated such that the minimum sum of the reward is -1 and the maximum sum of the reward is 1. The first condition checked for is failure. Failure is true only if the reaction wheels exceed their maximum speed or if the batteries are empty. Again, no data buffer constraints are considered. If failure does not occur, the imaging mode is checked next. If the satellite images the next upcoming target with the correct sensor type, a small reward bonus is returned. This reward bonus is equal to one 1 divided by the product of the total number of targets (50 targets per 90 decision-making intervals) and the summation of 1 and the square of the attitude error. The component due to imaging has a maximum reward of 1, which assumes the satellite images every target with the correct sensor type and zero attitude error. In this problem, the limiting factor is the number of targets in T as opposed to the number of decision-making intervals, so that's why the positive reward component is normalized by $|T|$.

$$R(s_i, a_i, s_{i+1}) = \begin{cases} -1 & \text{if failure} \\ \\ \dfrac{1}{|T|(1+\epsilon_{\text{att}}^2)} & \text{if } el_{\text{sc}} > el_{\text{min}} \text{ and } a_i \text{ is preferred} \\ \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

5.2.2.4 Gymnasium Environment

The generative transition function $s_{i+1}, r_i \sim G(s_i, a_i)$ for the multi-sensor EOS scheduling problem is also modeled using a Basilisk simulation. The multi-sensor EOS simulator is largely the same as that of the simple EOS and SSAEO scheduling problems. However, no ground stations or on-board data systems are modeled.

5.3 Performance Comparisons

In this section, DQN (Algorithm 3), A2C (Algorithm 5), PPO (Algorithm 6), SPPO (Section 2.3.6), MCTS-Train (Algorithm 7), and the GA (Algorithm 8) are benchmarked in each environment and compared to one another on the basis of performance, performance variance, and wall clock time. DQN, A2C, PPO, and SPPO are all Stable-Baselines3[1] (SB3) implementations. Stable-Baselines3 is a Python package with a collection of reliable implementations of various reinforcement learning algorithms. Each SB3 algorithm utilizes multiprocessing, and each implementation is synchronous, meaning that each actor takes a step at the same time in parallel. If one environment fails and must be reset, all the other actors must wait for the reset environment to initialize.

The first step of the performance comparison is to parameterize MCTS-Train and the genetic algorithm to ensure each of the algorithms are performing well given the available computational resources. These results can also provide baseline performance metrics for the reinforcement learning algorithms. Chapter 4 and past work tunes the hyperparameters regarding the activation function, number of training epochs, learning rate, batch size, dropout, loss function, and optimizer for the supervised learning portion of MCTS-Train [69, 64]. The optimized hyperparameters are fixed in each MCTS-Train experiment and are summarized in Table 5.1. The Leaky ReLU activation function is used for each hidden layer. The activation function is defined as follows:

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{5.6}$$

Each network is trained for a maximum of 10,000 epochs with a batch size of 45,000 to ensure stable convergence. The Adam optimizer with an initial learning rate of 1e-3 is utilized. The mean squared error loss function is used to train each network. Furthermore, a small amount of dropout is added to each hidden layer to help prevent overfitting. The probability of dropout is 0.01.

The hyperparameters of each SB3 algorithm (PPO, SPPO, A2C, and DQN) are tuned using a two-step process. In the first step, parameters such as the batch size, number of steps before an

[1] https://stable-baselines3.readthedocs.io/en/master/guide/algos.html

Table 5.1: Optimized hyperparameters for MCTS-Train.

Hyperparameter	Value
Activation Function	Leaky ReLU
α	0.1
Number of Training Epochs	1e4
Learning Rate	1e-3
Batch Size	4.5e4
Loss Function	Mean Squared Error
Optimizer	Adam
Dropout	0.01

update, number of training epochs, etc. are optimized for a network with 4 hidden layers and 20 nodes per hidden layer. Each algorithm utilizes the Leaky ReLU activation function with the same α parameter in Table 5.1. However, no dropout is utilized for these algorithms. In the second step, these optimized hyperparameters are deployed in a search over the number of hidden layers and the number of nodes per hidden layer. For PPO and shielded PPO, only one policy is trained for each hyperparameter combination because the algorithms are relatively stable. For A2C and DQN, five policies are trained and evaluated because the performance of the algorithms can vary widely between different seeds.

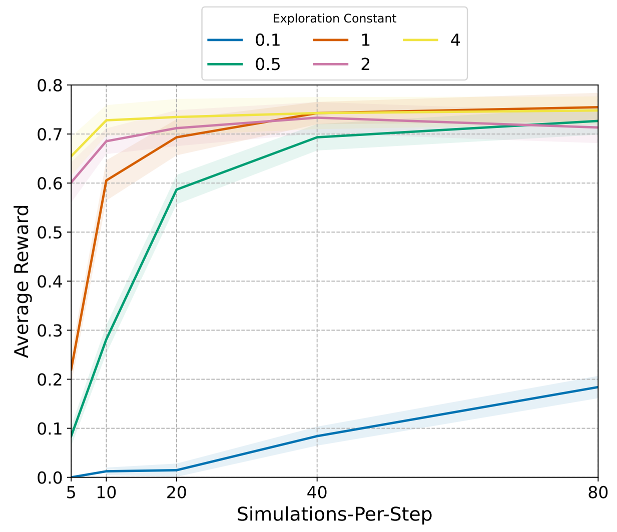
After the benchmarks are presented over the number of hidden layers and nodes for MCTS-Train and the SB3 algorithms, the optimized hyperparameters are selected for each algorithm and the algorithms are benchmarked once more. Five trials of training are performed for each algorithm. The training curves with variance between trials are evaluated to determine how quickly each algorithm converges and how much the algorithms can vary in performance between seeds.

5.3.1 MCTS Hyperparameter Tuning

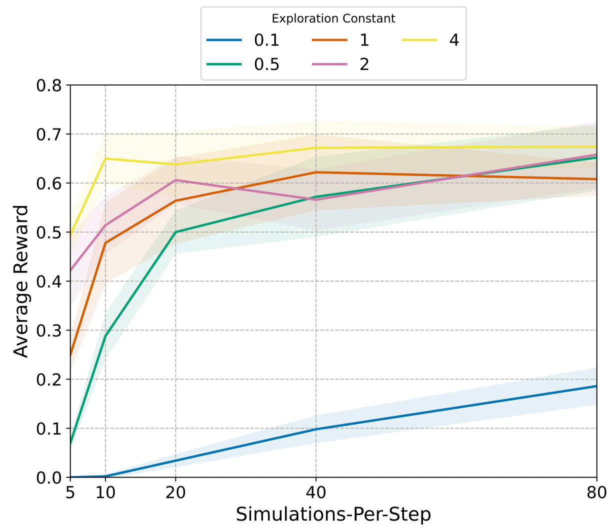
Before MCTS can be used to generate training data, the hyperparameters of MCTS must be tuned. The two tunable hyperparameters of MCTS are the number of simulations-per-step and the exploration constant. The number of simulations-per-step is the number of simulations MCTS runs per step through the environment. Generally speaking, the more simulations-per-step, the better the state-action value estimate. The exploration constant is used to scale the exploration term in MCTS, which is the square root of the log of the number of times a state has been visited divided by the number of times the state-action pair has been visited. The exploration constant is used to balance exploration and exploitation. The higher the exploration constant, the more that exploration is favored.

For each EOS scheduling problem, each combination of [5, 10, 20, 40, 80] simulations-per-step and [0.1, 0.5, 1, 2, 4] exploration constant are deployed for 30 trials. The results of the hyperparameter search are shown in Figure 5.1, where the solid line represents the average reward across the 30 trials and the transparent highlighting represents the 95% confidence intervals. For each EOS scheduling problem, higher exploration constants result in higher rewards. The number of simulations-per-step also has a significant effect on performance, but after a certain number there are depreciating returns for adding more simulations. This is especially true for the multi-sensor EOS scheduling problem with 45 decision-making intervals and the agile EOS scheduling problem, which also has 45 decision-making intervals. Simulations beyond 20 or so simulations-per-steps result in little performance improvement. For the multi-sensor EOS scheduling problem with 90 decision-making intervals, this is not necessarily true. The large confidence interval bounds and spread between the exploration constants suggests that even more simulations-per-step could be beneficial. However, the computational cost of running MCTS increases linearly with the number of simulations-per-step, so additional steps are not added.

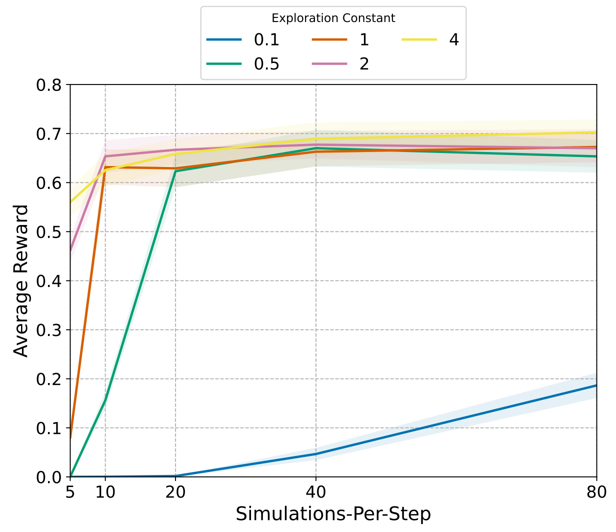
The results in Figure 5.1 also provide an indication for the performance that can be expected from the RL algorithms. The multi-sensor EOS scheduling problem with 45 decision-making intervals appears to have a maximum reward between 0.7 - 0.75. The multi-sensor EOS scheduling problem with 90 decision-making intervals appears to have a maximum reward between 0.6 - 0.7. The same is true for the agile EOS scheduling problem.



(a) Multi-sensor EOS scheduling problem with 45 decision-making intervals.



(b) Multi-sensor EOS scheduling problem with 90 decision-making intervals.



(c) Agile EOS scheduling problem.

Figure 5.1: MCTS hyperparameter search results: average reward with 95% confidence intervals.

5.3.2 Genetic Algorithm Hyperparameter Tuning

The genetic algorithm in Algorithm 8 is implemented for each EOS scheduling problem and benchmarked for various population sizes and generations to ensure the GA is correctly parameterized. A large enough population size is required to increase the probability to genetic algorithm

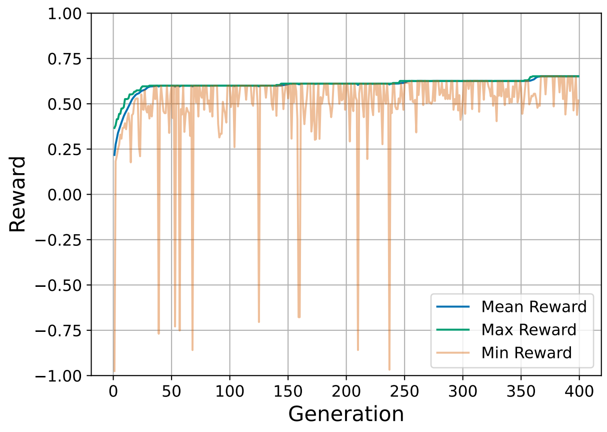


Figure 5.2: Example convergence curve for the genetic algorithm.

will start with decent individuals that can then be evolved over many generations to produce better offspring. These hyperparameter searches are presented alongside the benchmarks of the RL algorithms in Figures 5.3f, 5.4f, and 5.5f and discussed in relation to the RL benchmarks in the corresponding sections. In comparison to the MCTS benchmarks in Figure 5.1, the GA is able to match the performance for the Multi-Sensor EOS scheduling problem. The GA performs slightly worse than MCTS for the Agile EOS scheduling problem for the number of generations and population sizes explored. More computation in the form of larger population sizes and more generations could close the performance gap, but at a computational cost.

An example convergence curve for the genetic algorithm is provided in Figure 5.2. In this curve, a genetic algorithm with a population size of 400 is run for 400 generations on the SSAEO scheduling problem. The genetic algorithm converges to a good solution in the first 50 generations, largely due to the large population size. The genetic algorithm then continues to improve the solution over the next 350 generations, but the improvement in average reward is marginal.

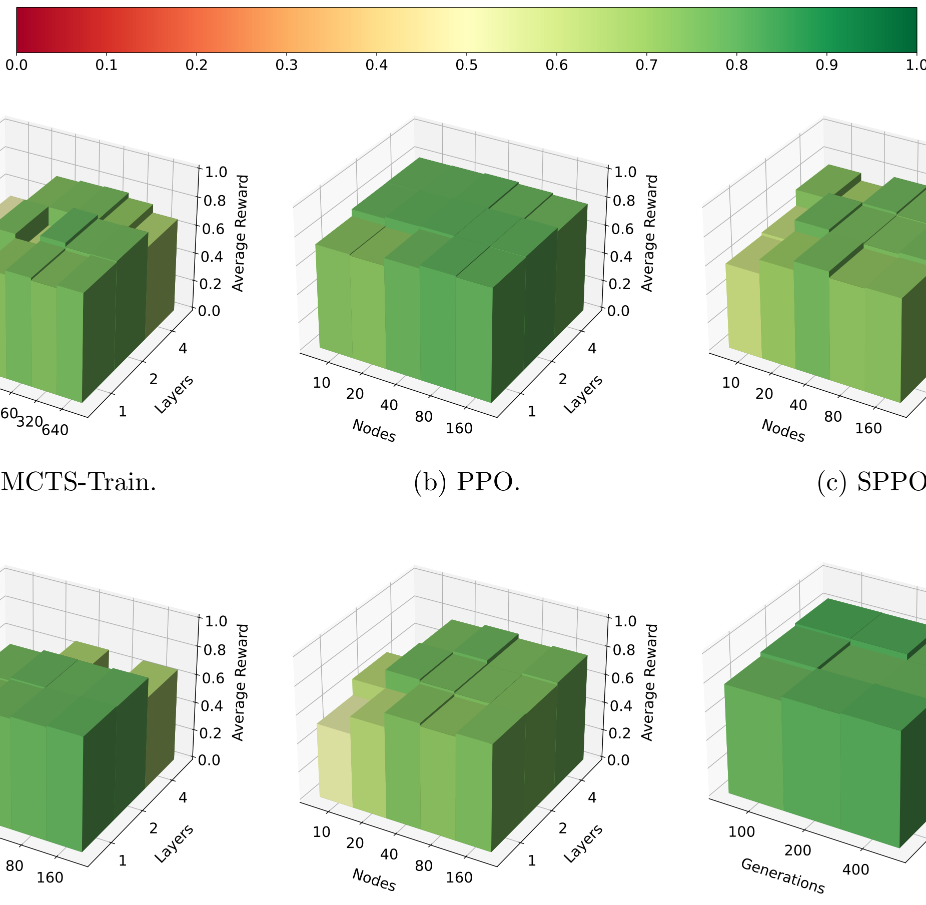
5.3.3 Multi-Sensor EOS Scheduling Problem Benchmarks

Each algorithm discussed thus far is benchmarked for two versions of the multi-sensor EOS scheduling problem, one with 45 decision-making intervals ($|I| = 45$) and one with 90 decision-making intervals ($|I| = 90$). In the former case, there are 25 total targets in the set T. In the latter case, there are 50 total targets in the set T. Multiple decision-making intervals are evaluated to determine the effect on the trained policies for each algorithm.

5.3.3.1 45 Decision-Making Intervals

The results for the hyperparameter searches of each algorithm are presented in Figure 5.3. MCTS-Train is benchmarked for networks with [1, 2, 4] hidden layers and [10, 20, 40, 80, 160, 320, 640] nodes per hidden layer. The SB3 algorithms (PPO, SPPO, A2C, and DQN) are benchmarked for networks with [1, 2, 4] hidden layers and [10, 20, 40, 80, 160] nodes per hidden layer because performance does not tend to improve outside these more limited parameters. Conversely, MCTS-Train sometimes benefits from few hidden layers but many nodes per hidden layer. Finally, the GA is benchmarked for [100, 200, 400] generations and population sizes of [100, 200, 400]. This range of parameters is sufficient to provide a good benchmark in a reasonable amount of time.

After each policy is trained for MCTS-Train and the SB3 algorithms, the performance of each is evaluated in the environment with approximately 150 trials. The genetic algorithm hyperparameter search is performed with only 20 trials for each hyperparameter combination. In this case, the genetic algorithm provides the upper bound on performance at around 0.8 average reward across all numbers of generations and population size. Each of the other algorithms is able to compute policies that are close to this upper bound. PPO is shown to be extremely stable over the entire range of network size hyperparameters selected and produces the best performance of all the RL algorithms across the board. Shielded PPO is shown to be slightly less stable and performant than PPO, but produces a policy that is close to the upper bound that never violates the resource constraints due to the shield. PPO itself is not guaranteed to never produce resource constraint violations, but it is easier for the algorithm to find high-performing policies because it is not constrained to a subset of the state space. MCTS-Train, DQN, and A2C are typically able to produce good performing policies, but are shown to be less stable than PPO and shielded PPO. Furthermore, MCTS-Train requires much larger network sizes to produce high performing policies. The maximum number of nodes evaluated for MCTS-Train is 640, compared to just 160 from the



(a) MCTS-Train.

(b) PPO.

(c) SPPO.

(d) A2C.

(e) DQN.

(f) GA.

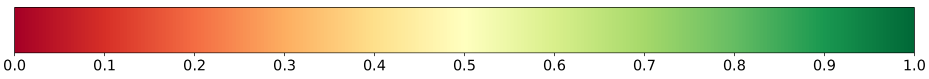
Figure 5.3: Multi-sensor EOS scheduling problem, $|I| = 45$. The average reward for each hyperparameter combination is displayed for each algorithm. Reward < 0 clipped for readability. **(A)** MCTS-Train average reward, hyperparameters: Table 5.1. **(B)** PPO average reward, hyperparameters: LR = 3e-4, batch size = 2070, epochs = 50. **(C)** SPPO average reward, hyperparameters: LR = 3e-4, batch size = 2070, epochs = 50. **(D)** A2C average reward, hyperparameters: LR = 7e-3, steps before update = 22. **(E)** DQN average reward, hyperparameters: LR = 1e-4, batch size = 2070, buffer size = 5e4. **(F)** GA average reward, hyperparameters: mutation rate = 0.25, crossover probability = 0.25, tournament size = 3.

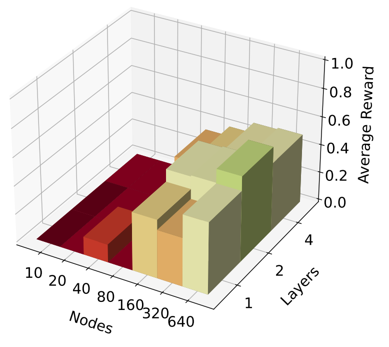
other algorithms. At face value, this study would indicate that PPO is marginally better than the other RL algorithms. However, the number of decision-making intervals utilized in this experiment is relatively small. The number of decision-making intervals is doubled in the next experiment to determine if the performance of the algorithms changes.

5.3.3.2 90 Decision-Making Intervals

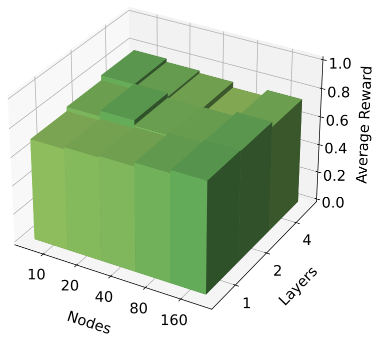
An identical experiment is performed for the multi-sensor EOS scheduling problem with 90 decision-making intervals. This experiment is performed to determine how the size of the search space impacts performance. The hyperparameters swept over in this experiment are the exact same as those in the previous experiment for each algorithm. However, there are slight differences in the hyperparameters regarding batch size, epochs, etc. The results of this search are presented in Figure 5.4.

In this case, both the genetic algorithm and MCTS-Train are capable of producing good performance, but in general struggle to match the performance of the other algorithms. MCTS-Train is only able to find one policy that achieves more than 0.5 average reward. The reason for this is that both of these algorithms are searching over the action space to find optimal policies. The total number of possible trajectories through the environment is $|\mathcal{A}|^{|I|}$. 90 decision-making intervals results in vastly more possible trajectories, which makes it more difficult for these algorithms to find good solutions. This indicates that the genetic algorithm and MCTS-Train may not be well suited if a long planning horizon is desired for training, i.e. training for a full day or a full week of operations. PPO, SPPO, and A2C are shown to be the most stable and performant algorithms. They are all able to produce good performing policies as they are optimizing over the state space instead. Furthermore, these are all policy gradient algorithms, which might also help explain why they perform better. DQN's performance is more on par with the genetic algorithm or the best case scenario for MCTS-Train, but is certainly worse than the performance of DQN on the 45 decision-making interval version of the multi-sensor EOS scheduling problem. Regardless of the degradation in performance, DQN is fairly stable over the selected hyperparameters, which MCTS-Train is not. MCTS-Train is likely unstable for this problem because MCTS has not properly converged, as demonstrated by the large confidence intervals and spreads between hyperparameters in Figure 5.1b. Again, this indicates that MCTS-Train may not work well for long planning horizons. PPO, SPPO, A2C, and DQN could all be trained on even longer decision-making intervals without much

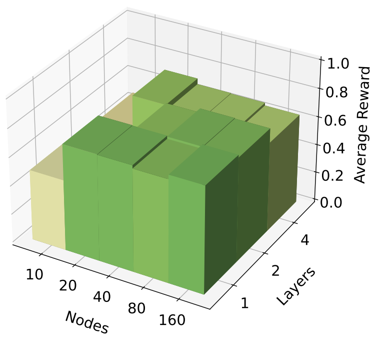




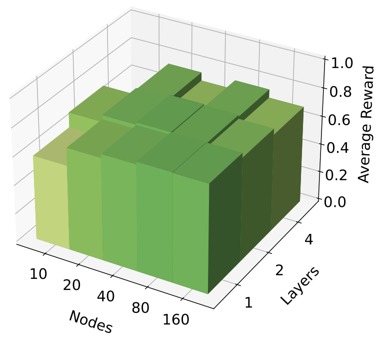
(a) MCTS-Train.



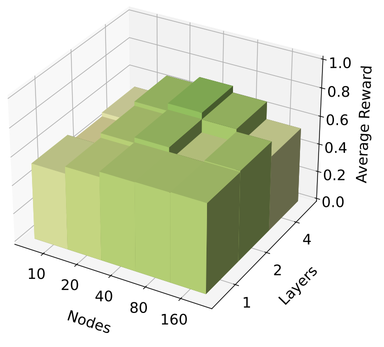
(b) PPO.



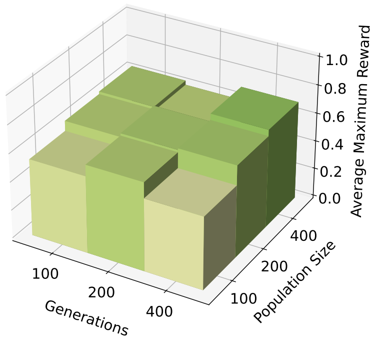
(c) SPPO.



(d) A2C.



(e) DQN.



(f) GA.

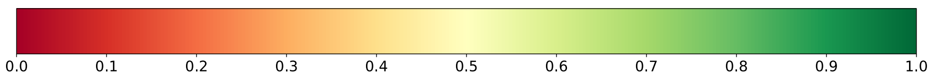
Figure 5.4: Multi-sensor EOS scheduling problem, $|I| = 90$. Average reward for the selected hyperparameters of each algorithm. Reward < 0 clipped for readability. **(A)** MCTS-Train average reward, hyperparameters: Table 5.1. **(B)** PPO average reward, hyperparameters: LR = 3e-4, batch size = 4140, epochs = 50. **(C)** SPPO average reward, hyperparameters: LR = 3e-4, batch size = 1040, epochs = 50. **(D)** A2C average reward, hyperparameters: LR = 7e-3, steps before update = 45. **(E)** DQN average reward, hyperparameters: LR = 1e-4, batch size = 2070, buffer size = 5e4. **(F)** GA average reward, hyperparameters: mutation rate = 0.25, crossover probability = 0.25, tournament size = 3.

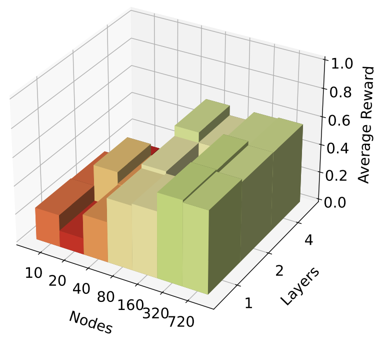
increase in computational complexity provided that the same number of steps are used for each. However, DQN is likely not a good candidate as the number of decision-making intervals increases, as evidenced by the reduction in performance in the 90 decision-making interval version of the multi-sensor EOS scheduling problem.

5.3.4 SSAEO Scheduling Problem Benchmarks

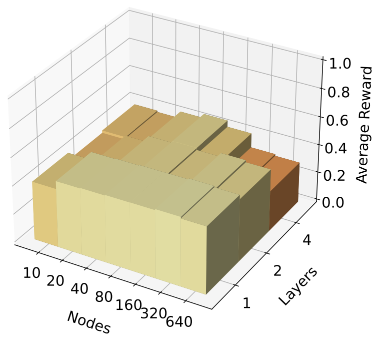
A final hyperparameter experiment is performed for the agile EOS scheduling problem. The results are presented in Figure 5.5. The same MCTS-Train and genetic algorithm hyperparameter combinations used in the last two experiments are again used in this one. However, the sizes of the neural networks for the SB3 algorithms are increased to ensure a large enough search space is explored. The SB3 algorithms are benchmarked for [1, 2, 4] hidden layers and [10, 20, 40, 80, 160, 320, 640] nodes per hidden layer.

In this case, only MCTS-Train and the genetic algorithm are able to produce high-performing policies. Each of the other algorithms converges to some locally maximal policy. PPO, SPPO, and A2C are shown to be the most stable, while DQN is shown to have large instability and poor robustness for neural networks with more than 160 nodes per hidden layer. The reason for the relatively poor performance of the SB3 algorithms is likely due to the fact that the agile EOS scheduling problem is more complex than the multi-sensor EOS scheduling problem in terms of resource management and science objectives. The agile EOS scheduling problem has a more complex reward function, an additional resource constraint, more sparse reward, and a larger action space. MCTS-Train is able to leverage its high quality rollout policy, which initially finds a safe trajectory of actions that MCTS can improve upon. The state-action value estimates produced by MCTS are closer to the optimal state-action value function, which are then regressed over using the artificial neural networks. While a similar shield is used within shielded PPO, shielded PPO ultimately bounds the decision-making agent's actions to the safety states. MCTS, however, allows the agent to explore actions that may violate these safety limits with exceeding the limits used within the reward function. When this difference in performance is compared to the multi-sensor EOS scheduling problem results, it is evident that MCTS-Train may be an excellent choice of algorithm as long as the number of decision-making intervals is not too large. A key question that remains, however, is how the EOS scheduling problems formulated in this thesis translate to real-world EOS scheduling problems with larger data buffers and less aggressive momentum buildup. If

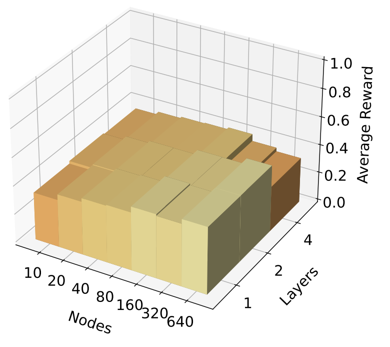




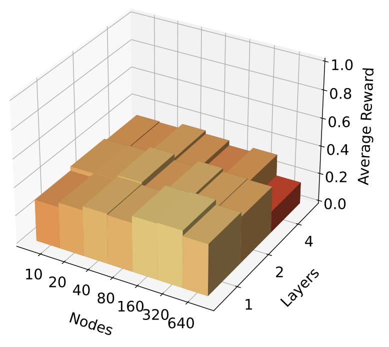
(a) MCTS-Train.



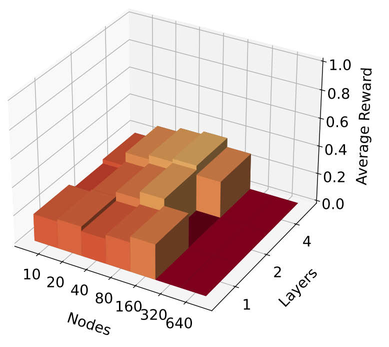
(b) PPO.



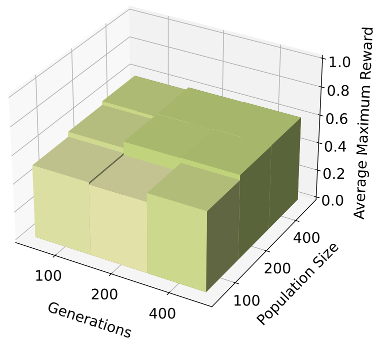
(c) SPPO.



(d) A2C.



(e) DQN.



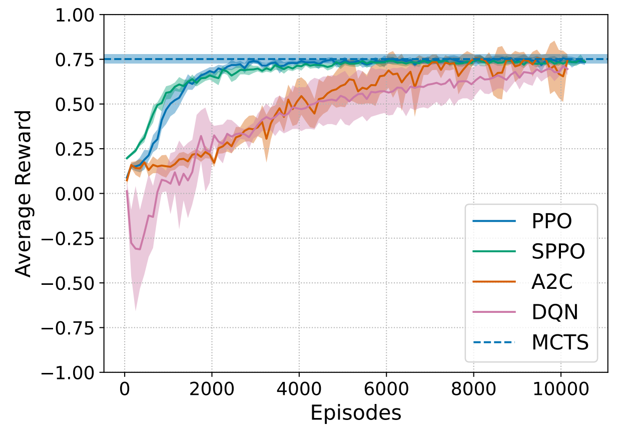
(f) GA.

Figure 5.5: Agile EOS scheduling problem. Average reward for the selected hyperparameters of each algorithm. Reward < 0 clipped for readability. **(A)** MCTS-Train average reward, hyperparameters: Table 5.1. **(B)** PPO average reward, hyperparameters: LR = 3e-4, batch size = 4140, epochs = 100. **(C)** SPPO average reward, hyperparameters: LR = 3e-4, batch size = 2070, epochs = 100. **(D)** A2C average reward, hyperparameters: LR = 3e-4, steps before update = 11. **(E)** DQN average reward, hyperparameters: LR = 1e-4, batch size = 2070, buffer size = 5e4. **(F)** GA average reward, hyperparameters: mutation rate = 0.25, crossover probability = 0.25, tournament size = 3.

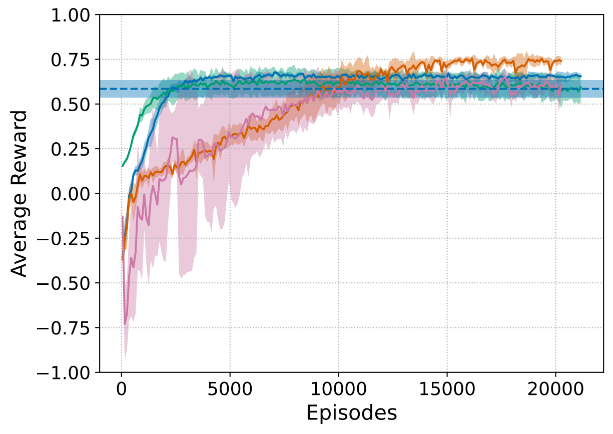
the problems can be scaled such that only 45 decision-making intervals are required to sufficiently model the problem, MCTS-Train may make an excellent choice of algorithm due to its ability to handle many resource constraints and complex science objectives.

5.3.5 Performance Variance

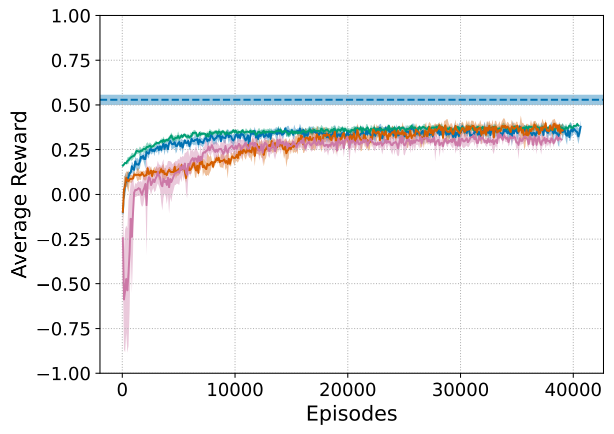
To evaluate the consistency of each algorithm between different training runs and initializations of the neural networks, a performance variance experiment is performed. For PPO, SPPO, A2C, and DQN the best performing hyperparameters in step one of the optimization process are deployed for five trials of training, each with different initial random seeds. For MCTS-Train, the best-performing network hyperparameter combination is selected for the five trials. The genetic algorithm is not considered in this experiment as the results from the GA are not generalized with a neural network. The average reward curves with 1σ standard deviation for each algorithm and each problem are plotted in Figure 5.6. PPO and SPPO are shown to produce the smallest standard deviation between policies and converge the quickest. This result is not necessarily surprising, as Reference [60] touts PPO as being a reliable algorithm needing little hyperparameter tuning. This claim is also backed up with the hyperparameter searches presented within this chapter that find that vanilla PPO is extremely stable across the range of hyperparameters. SPPO is not as stable as PPO over the entire range of hyperparameters, but is found to be very stable when a good hyperparameter combination is selected. This is likely due to the fact that it is more difficult to find a high-performing policy that avoids the limits of the safety MDP. A fundamental trade-off exists between resource utilization and science collection. In contrast to the other algorithms, A2C and DQN are shown to be more unstable both across the entire range of hyperparameters and between different runs of the same hyperparameters, particularly for <1e4 episodes. As a result, they take longer to converge. In fact, the maximum number of episodes in Figure 5.6 is primarily driven by these two algorithms. Finally, MCTS-Train is shown to have a very small variance between the different network seeds for the same hyperparameters. While MCTS-Train can be relatively unstable across the entire range of hyperparameters, it is on par with PPO and SPPO in terms of stability when optimized hyperparameters are utilized. When examining the performance variance of each algorithm in addition to the performance across the hyperparameters presented in the last section, it is evident that PPO, SPPO, and MCTS-Train should be among the top candidates for



(a) Multi-sensor EOS, $|I|$=45.



(b) Multi-sensor EOS, $|I|$=90.



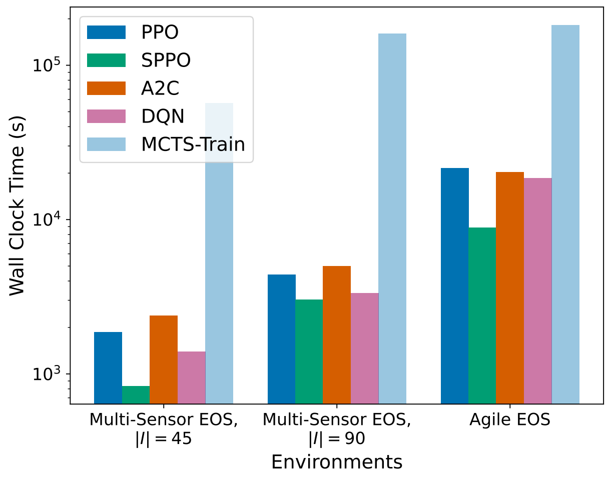
(c) Agile EOS.

Figure 5.6: Average reward and 1σ standard deviation across 5 trials using the optimized hyperparameters for each algorithm. PPO, SPPO, A2C, DQN: nodes = 20, hidden layers = 4. MCTS-Train utilizes the sizes of the best networks from Figures 5.3a, 5.4a, and 5.5a.

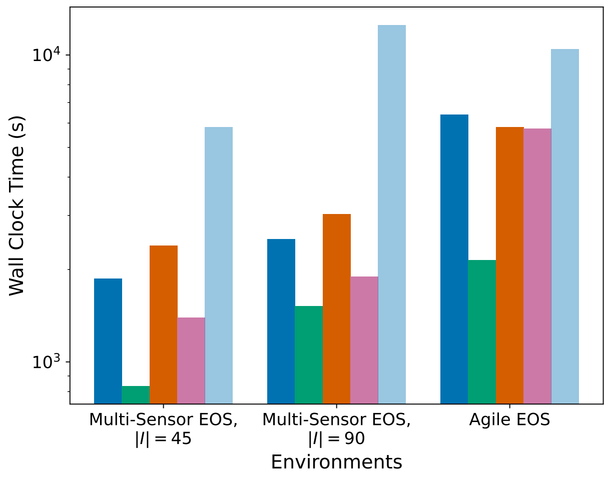
solving EOS scheduling problems. However, MCTS-Train should only be used in cases where the number of decision-making intervals is relatively small (i.e., $|I| \leq 45$).

5.3.6 Wall Clock Time

The wall clock times for each algorithm using the hyperparameters in Section 5.3.5 are displayed in Figure 5.7. The computational experiments resulting in these wall clock times were performed on an AMD 3960X Threadripper CPU with 64 GB of RAM and an NVIDIA 3070 graphics card. The wall clock times in Figure 5.7a are for the total number of episodes for each algorithm, and the wall clock times in Figure 5.7b are capped at 10,000 episodes. MCTS-Train is the most computationally expensive algorithm because of the amount of data generated and the number of neural networks produced during the training process. Furthermore, MCTS requires that the Basilisk simulation is rewound by stepping through the trajectory of past actions to create a new child node during the simulation step. This results in a lot of wasted computation. On average, half of the trajectory of executed actions during tree generation are wasted. If Basilisk simulations can be deep copied at the Python level in the future, which could be made possible by moving away from SWIG, then this computational performance could be drastically improved (likely at the cost of increased RAM utilization). Until this happens, MCTS-Train should only be used for complex problems that other algorithms struggle to generate high-performing policies for. PPO, A2C, and DQN are typically in the same ballpark computationally, with the relative performance dependent on the specific problem. In comparison to MCTS-Train, they are single trajectory algorithms that do not rely on building a search tree, so they do not suffer from the need to rewind the simulation in the same manner. SPPO is the least computationally expensive algorithm. The reason for this is that SPPO guarantees that resource constraint violations do not occur during execution. When a resource constraint violation occurs for any of the SB3 algorithms, the simulation is reset, meaning that an entire new target set must be generated by executing a Basilisk simulation and computing access times to construct the list of targets. This is an expensive procedure. Because the SB3 algorithms are synchronous implementations, all the workers must wait until the reset environment is ready to step through the environment again before they can begin stepping again. As a result, overall CPU utilization decreases and wall clock time increases. Therefore, future works may want to consider using asynchronous implementations of these algorithms. The risk of this is potentially giving up the performance and stability of the SB3 implementations.



(a) All episodes.



(b) 10,000 episodes.

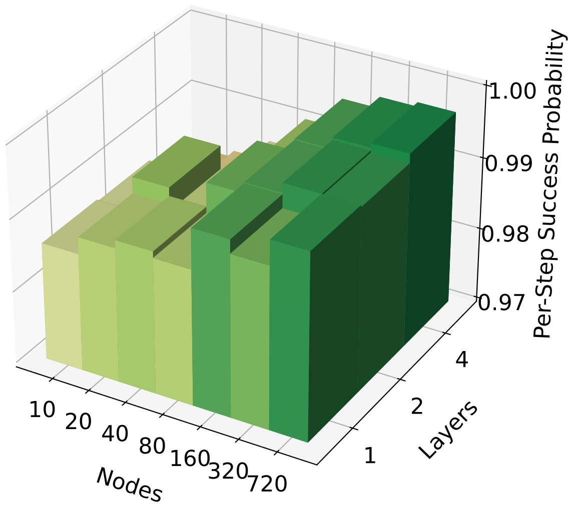
Figure 5.7: Wall clock times associated with each algorithm. Times are displayed for all episodes used in Figure 5.6 and for 10,000 episodes.

5.4 Long Duration Deployment

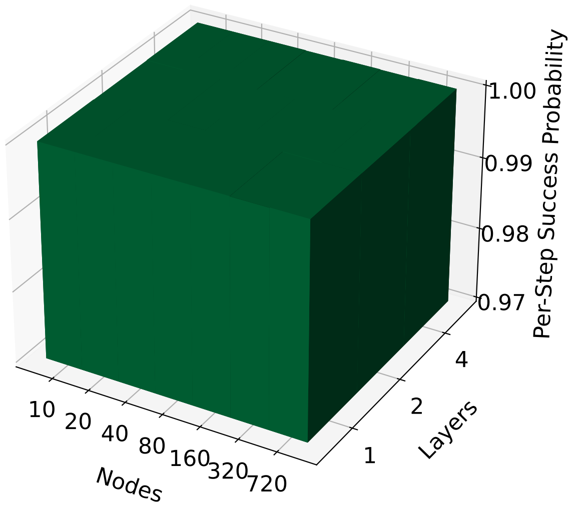
The performance comparisons provided in Section 5.3 provide insight into which algorithms are best suited for EOS scheduling problems in terms of average reward, performance variance, and wall clock time. However, all of these results are gathered for policies and value functions that are trained to operate in planning horizons that are between one and three orbits long. In a real operational scenario, a satellite will operate until it is decommissioned, which may take years. Therefore, it is important to understand how the performance of these algorithms changes as the planning horizon increases, particularly regarding safety. Furthermore, it's also important to understand the capabilities of each algorithm in terms of the length of planning horizon used in training. In its current configuration, MCTS-Train struggles to train high-performing policies for planning horizons with more than 45 decision-making intervals. On the other hand, PPO can be easily configured to train policies for planning horizons that are hundreds of decision-making intervals long. To address these questions, this section benchmarks the safety of policies trained by MCTS-Train and PPO. While the policies may be trained in different ways, each policy is deployed in the SSAEO environment for a day of operations. The total number of decision-making intervals increases from $|I| = 45$ to $|I| = 240$. The sum of the reward function still has a range of [-1, 1] because the relevant components are normalized by the total number of decision-making intervals. Lastly, the same target density of 135 targets per three orbits is assumed. Over the new planning horizon, each agent has 720 targets in the set T available for imaging.

The policies generated by MCTS-Train in Section 5.3 are deployed in day-long planning horizons in two configurations. The first configuration simply uses the policy in Equation (4.9) to select the next action. The second configuration wraps the MCTS-Train policy with the shield policy in Table 4.2 to guarantee safety during long-duration deployment. Nominal safety states result in the shield policy deferring to the action recommended by MCTS-Train. Off-nominal safety states result in the shield action. Finally, two training configurations for PPO are utilized. In the first configuration, the policies generated by PPO in Section 5.3 are deployed in day-long planning horizons. In the second training configuration, various PPO policies are trained to operate in day-long planning horizons, which MCTS-Train cannot be set up to do without exceeding the capabilities of the available computational resources. The policies trained in day-long planning horizons are then deployed in day-long planning horizons.

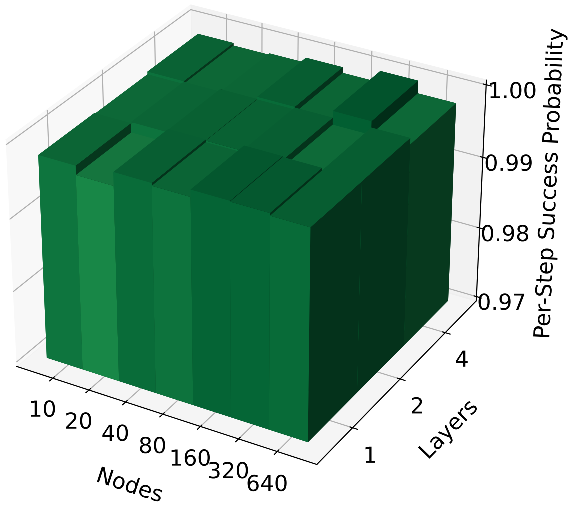
The performance of each deployment method may be found in Figure 5.8. The evaluated metric is the per-step probability of success. This is the probably that the policy will select an action at each step that does not violate a resource constraint. In Figure 5.8a, the performance of MCTS-Train deployed in the day-long planning horizon is displayed. Note that the z-axis begins at 0.97. MCTS-Train is able to produce policies that have at minimum a 98% chance of selecting an action that avoids failure and at maximum a 99.5% chance of selecting an action that avoids failure. While these probabilities may seem high, they typically result in a failure somewhere along the 240 decision-making intervals. In Figure 5.8b, the performance of the MCTS-Train policies wrapped in the safety shield are displayed. The safety shield ensures that a safe action is always taken, bringing the probability of success to 100%. The results for PPO are provided in Figures 5.8c and 5.8d. Compared to the unshielded MCTS-Train policies, PPO policies trained with 45 decision-



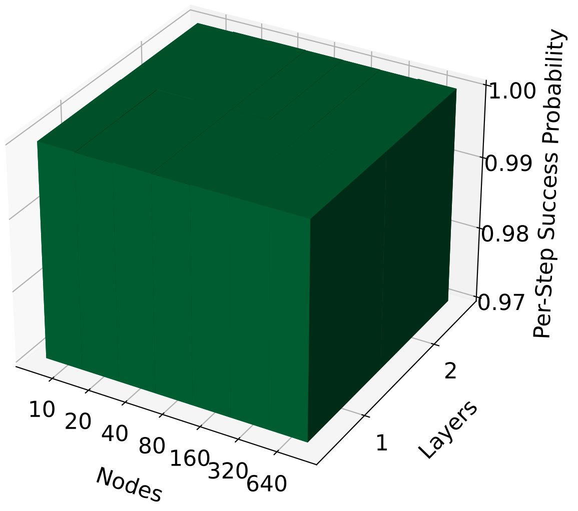
(a) MCTS-Train. Short duration training. Long duration unshielded deployment.



(b) MCTS-Train. Short duration training. Long duration shielded deployment.



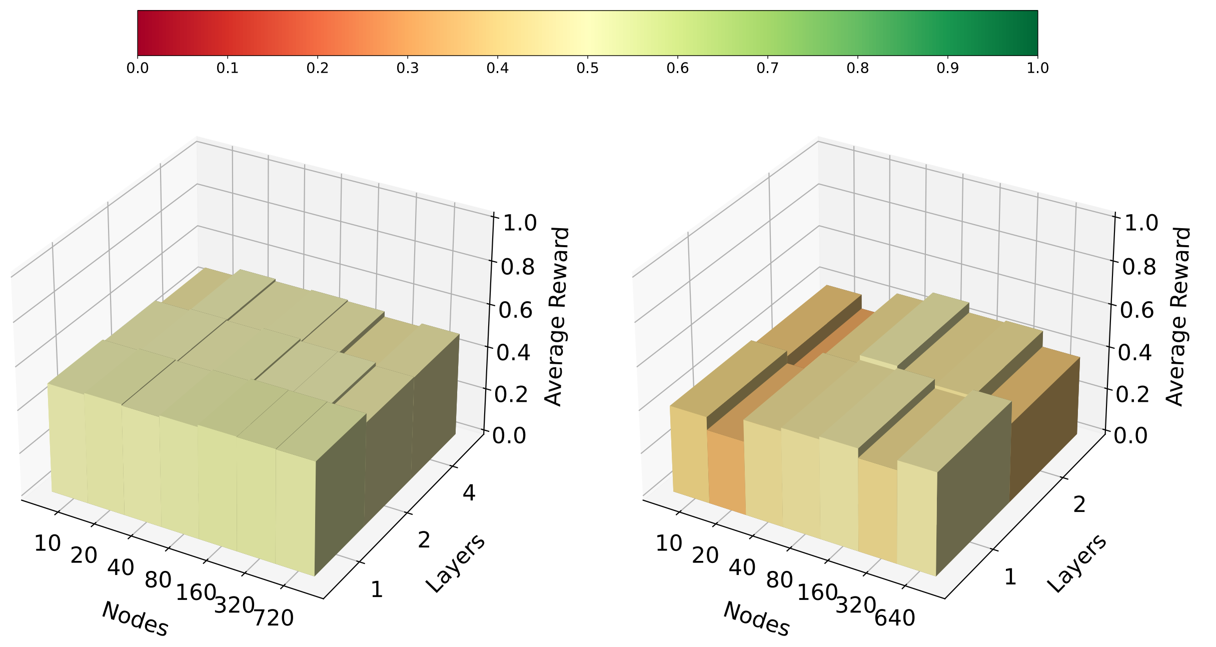
(c) PPO. Short duration training. Long duration deployment.



(d) PPO. Long duration training. Long duration deployment.

Figure 5.8: Per-step probability of success for each training and deployment method.

making intervals perform much better, with all policies achieving over a 99.7% chance of success at each step. In Figure 5.8d, the performance of the PPO policies trained with 240 decision-making intervals is displayed. Each of these policies achieves greater than a 99.99% probability of success.



(a) MCTS-Train. Short duration training. Long duration shielded deployment.

(b) PPO. Long duration training. Long duration deployment.

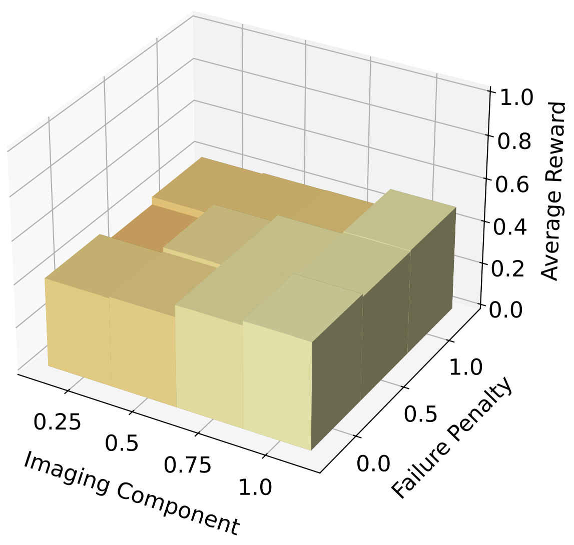
Figure 5.9: Average reward for each training and deployment method.

In Figure 5.9, the average reward for the shielded MCTS-Train policies and PPO policies trained for 240 decision-making intervals is displayed. The shielded MCTS-Train policies perform slightly better across the board. MCTS-Train solves for very greedy policies that attempt to collect and downlink as many images as possible. As evidenced by Figure 5.1c, there is a lot more reward possible in the environment if the approximated policy or value function can capture all of the nuances of the environment. MCTS-Train attempts to, but the errors in the value function manifest in the form of worse safety guarantees if shielding is not utilized. The shielding bounds the decision-making agents' greedy behavior, resulting in high-performing polices that respect the safety constraints.

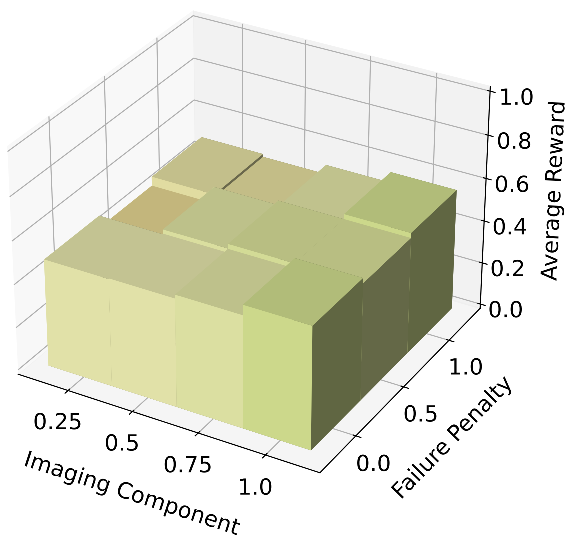
5.5 Reward Function Engineering

The last question this chapter addresses is how the reward function for EOS scheduling problems should be engineered, specifically the SSAEO scheduling problem, which is the most challenging of the problems explored thus far. The reward functions in Equations 3.7 and 5.1 mathematically formalize the objectives of the SSAEO scheduling problem. In the first iteration of the SSAEO scheduling problem reward function, Equation 3.7, the failure penalty is set to -10. The agent receives $0.1/p_j$ reward for every new target imaged and $1/p_j$ reward for every new target downlinked. The minimum reward possible is -10, and the maximum reward is shown to be around 29 (Figure 4.9). In this chapter, the reward function is modified slightly to ensure that minimum reward is no less than -1 and the maximum reward is no more than 1. The failure penalty is adjusted accordingly, the imaging (B = 0.1) and downlink (A = 0.9) constants are adjusted from their original values of 0.1 and 1, and the image and downlink reward components are both divided by the maximum number of decision-making intervals ($|I| = 45$). The first question to address is how these changes to the minimum and maximum reward impact the learned policies, if at all.

The second question to address is why the failure penalty, downlink constant, and imaging constant are set as they are. The current formulation of the reward function heavily prioritizes downlink, which can make reward more sparse. Furthermore, the failure penalty is quite large in comparison to the maximum possible reward. In Chapter 4, the maximum reward achieved is almost three times larger than the failure penalty. In this chapter, the maximum reward is half of the failure penalty. How does this change effect the learned policies? Finally, can the reward function be engineered to be more friendly to algorithms like PPO? PPO offers excellent convergence properties, is shown to be stable and robust across a range of hyperparameters, and can be trained in larger planning horizons with ease. However, it seems to have a tendency to get stuck in a local minimum. Can this be addressed with a slight modification to the reward function?



(a) Stochastic policy.



(b) Deterministic policy.

Figure 5.10: Deterministic vs. stochastic policy for the reward function engineering experiment using PPO. Hidden layers = 1. Nodes = 640. Batch size = 517. Training epochs = 50. Training episodes = 10,000. Validation episodes = 100.

5.5.1 PPO Reward Engineering

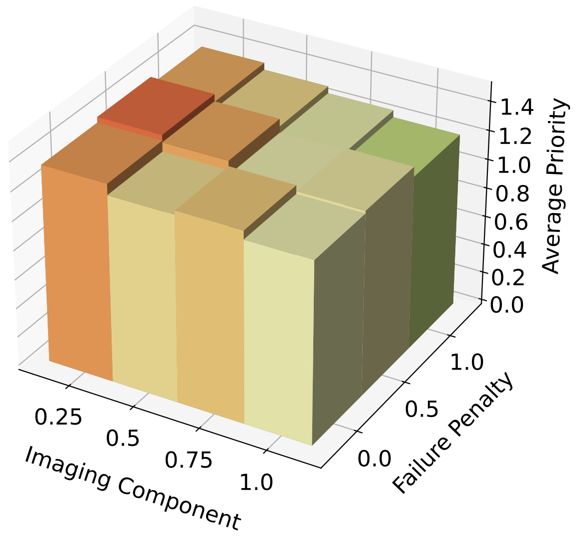
To begin answering this question, a search over the parameters in the reward function in Equation 5.1 is conducted. A policy is trained using PPO for each combination. The following imaging constants, $B = [0.25, 0.5, 0.75, 1.0]$, are utilized. The downlink constant A is calculated as $A = 1 - B$ to ensure that the maximum possible reward is equal to 1. Finally, failure penalties of [0, 0.5, 1] are applied for each imaging constant. In Figure 5.10b, the average reward for each combination is displayed. Two types of policy execution are explored: deterministic and stochastic. In the deterministic case, the action with the highest probability is selected. In the stochastic case, the action is sampled from the probability distribution. The average reward for the stochastic case is displayed in Figure 5.10a, and the average reward for the deterministic case is displayed in Figure 5.10b. The deterministic policy outperforms the stochastic policy, which makes sense because the cost of violating a resource constraint is large.

The first observation over the reward function sweep is that the failure penalty does not appear to have a large impact on performance. The reason for this is likely due to the fact that the decision-making agent is still incentivized not to fail in order to avoid early episode termination, which will reduce the maximum possible reward possible. The second observation regarding the reward function sweep is that reward tends to increase as the imaging is weighted more heavily over downlink. However, it's difficult to tell from Figure 5.10 whether this has a meaningful impact on the behavior of the learned policy.

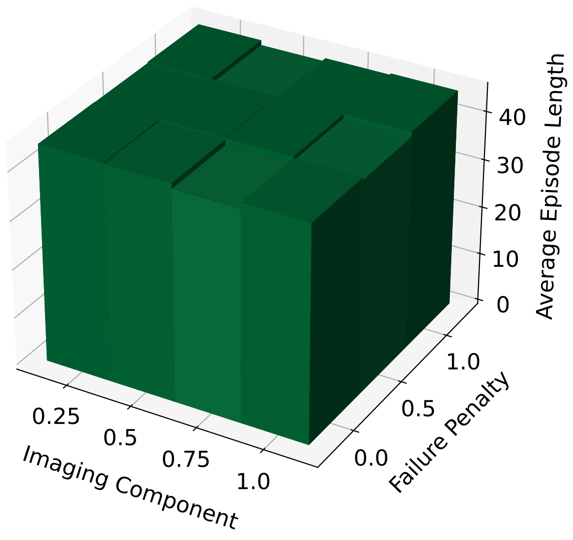
To explore this in greater detail, the average priority of each image, average simulation length, average number of imaged targets, and average number of downlinked targets are plotted in Figure 5.11. The average simulation length, average number of images, and average number of downlinked targets are roughly the same over the parameters. However, the average priority of each image gets lower as the imaging component gets larger. Recall that a target with a priority of 1 is the "highest" priority target possible in that it leads to the largest reward bonus because the reward is formulated as $1/p_j$. Therefore, lower numerical priorities on average are preferred. As reward is more heavily weighted towards imaging, reward becomes far less sparse, and PPO is able to infer which priorities are the most worth imaging. This is the best explanation for the increase in reward in Figure 5.10.

5.5.2 MCTS-Train Reward Engineering

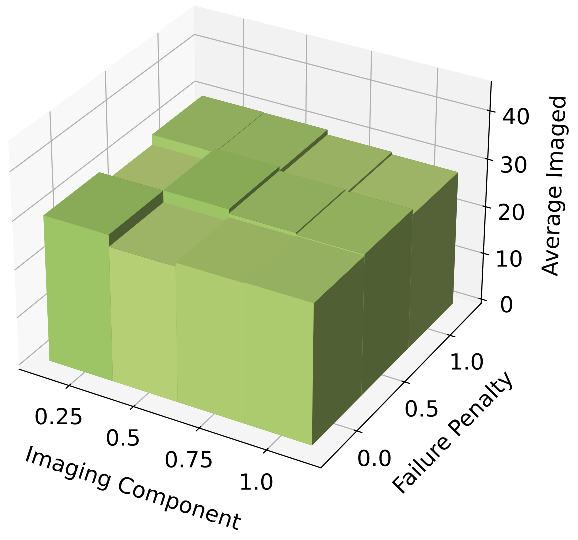
While the PPO reward engineering experiments present interesting results for PPO, they may not hold up for MCTS-Train, which is a much different algorithm. Furthermore, the question of whether or not the transition from Equation 3.7 to Equation 5.1 for the reward function leads to any appreciable difference in metrics regarding the average number of targets imaged and downlinked remains. Of most interest is whether a larger reward penalty fundamentally changes the learned policy. Furthermore, closer examination of the policies generated by MCTS-Train may help explain why PPO performs suboptimally for the SSAEO scheduling problem. To address these questions, the MCTS-Train policies presented in Figure 5.5a are examined in more detail. Furthermore, an



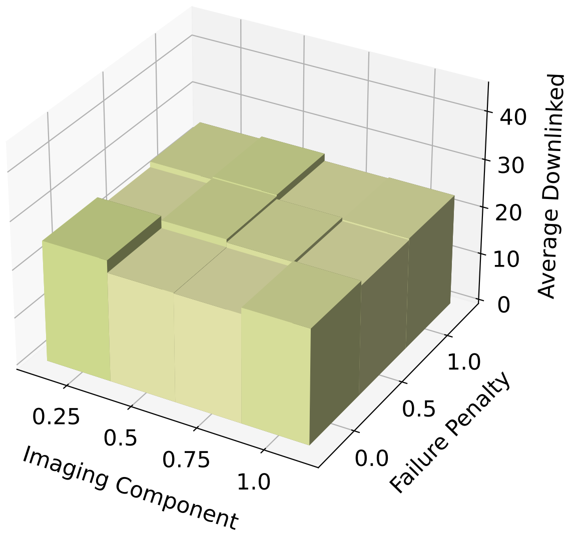
(a) Average priority.



(b) Average simulation length.



(c) Average number of images.



(d) Average number of downlinked images.

Figure 5.11: Reward function engineering experiment using deterministic policy trained with PPO. Hidden layers = 1. Nodes = 640. Batch size = 517. Training epochs = 50. Training episodes = 10,000. Validation episodes = 100.

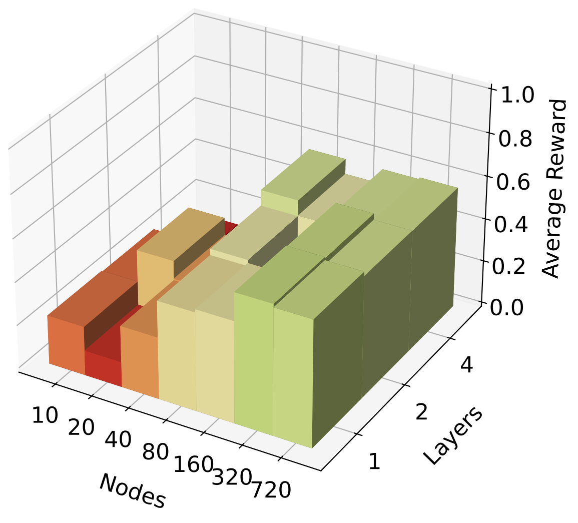
additional MCTS-Train experiment with $A = 0.9$, $B = 0.1$, and $F = 0.25$ is performed, and the results of this experiment are examined in detail.

In Figure 5.12, the average reward, success rate, average number of imaged targets, and average number of downlinked targets are plotted for the SSAEO scheduling problem with a downlink component of 0.9, imaging component of 0.1, and failure penalty of 1. In comparison to Figure 4.9, which corresponds to the reward function in Equation 3.7, the average number of imaged and downlinked targets is slightly lower. In Figure 4.9, the learned policies achieve 34 images collected on average and 30 images downlinked on average. In Figure 5.12, the highest-performing agents achieve 2-3 images collected and downlinked lower on average. This difference is likely not due to the difference in failure penalty relative to the upper bound on positive reward, but instead due to slight changes in the simulator regarding the desaturation mode.

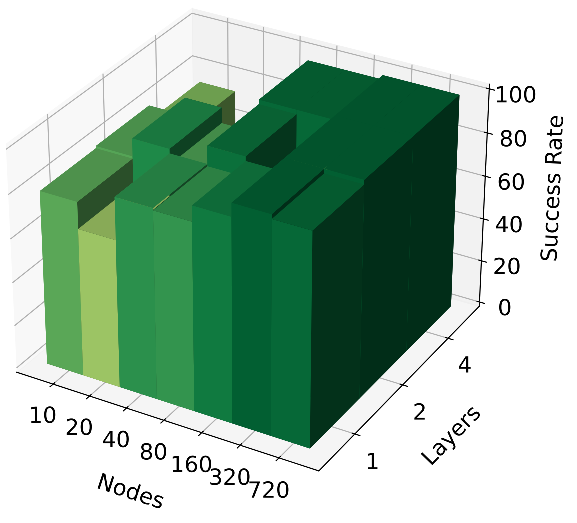
In Figure 5.13, MCTS-Train is applied to the SSAEO scheduling problem with the following reward components: $A = 0.9$, $B = 0.1$, and $F = 0.25$. The average reward of the learned policy is quite a bit lower than that of the learned policies in Figure 5.12. This is partially due to a few more failures in the highest performing policies and an overall reduction in the average number of downlinked targets. For the high-performing policies, the average number of downlinked targets is far lower. A possible reason for this is that the larger failure penalty incentivizes the decision-making agent to prefer downlink over imaging in many cases in order to avoid resource constraint violations due to buffer overflows. The lower failure penalty agent tends to collect more images than the higher failure penalty agent, which also supports this argument.

5.6 Conclusion

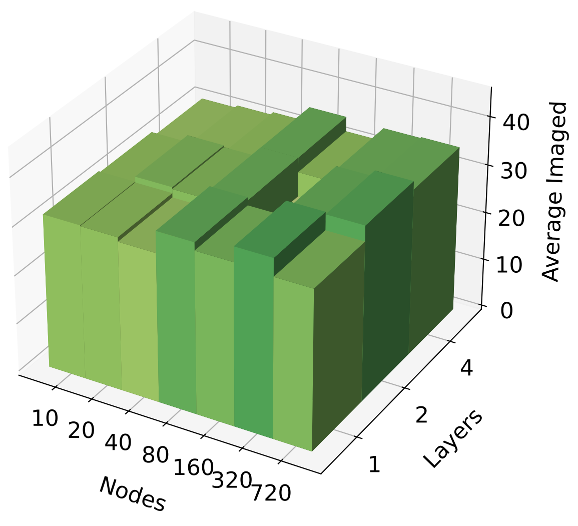
This chapter compares MCTS-Train, several SB3 implementations of state-of-the-art deep reinforcement learning algorithms, and a genetic algorithm for two EOS scheduling problems on the basis of performance, performance variance, wall clock time, and long duration deployment. MCTS-Train is shown to meet or exceed the performance of the SB3 DRL algorithms for small numbers of decision-making intervals, $|I| \leq 45$. However, for larger numbers of decision-making intervals, the SB3 algorithms are shown to outperform MCTS-Train. MCTS-Train is also the most computationally expensive algorithm, not including the genetic algorithm. Of the SB3 algorithms,



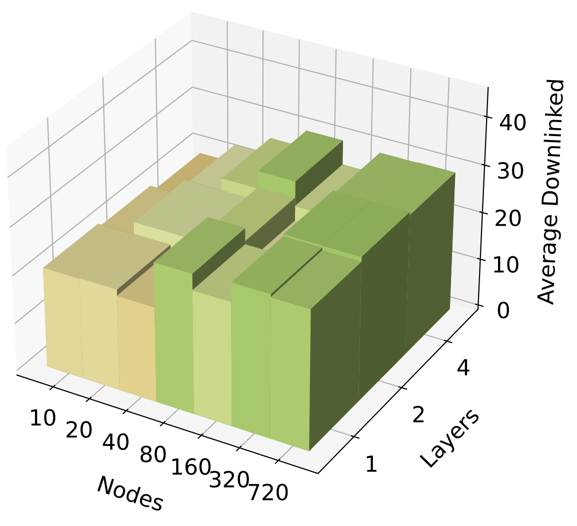
(a) Average reward.



(b) Success rate.



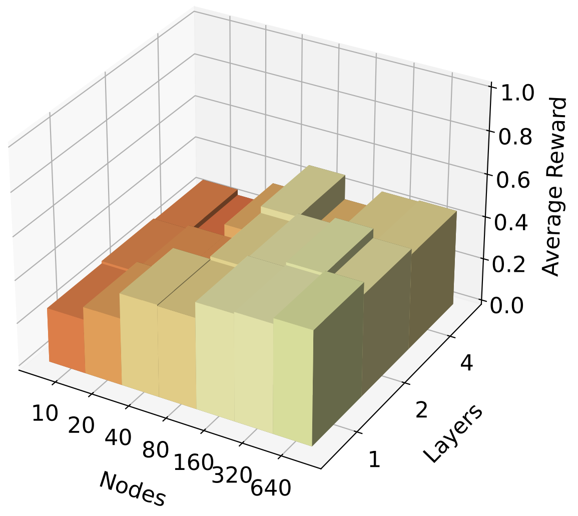
(c) Average number of imaged targets.



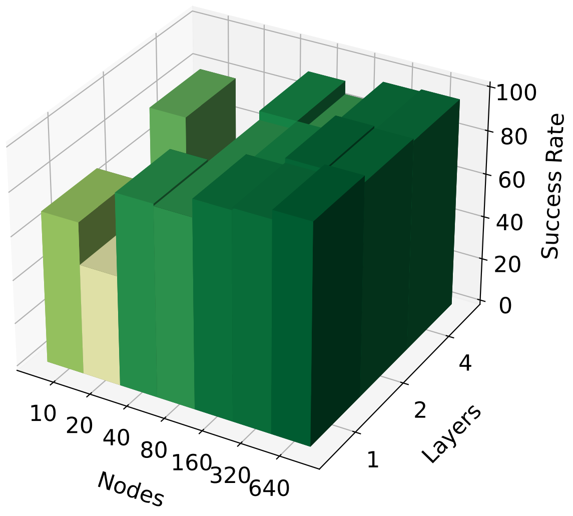
(d) Average number of downlinked targets.

Figure 5.12: MCTS-Train policy metrics. Reward components: $A = 0.9$, $B = 0.1$, $F = 1$.

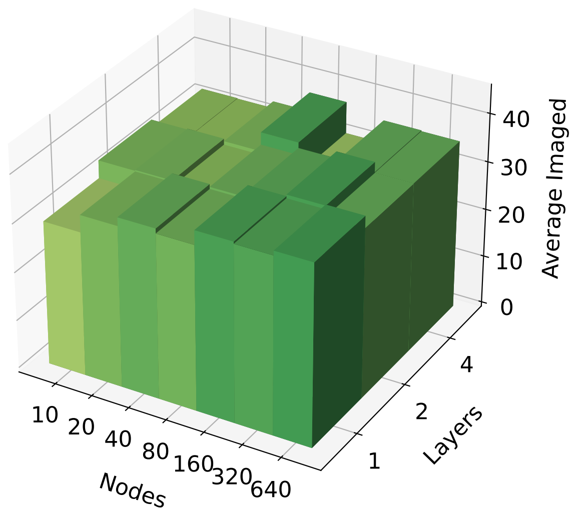
PPO is shown to be the most stable and quick to converge. Shielded PPO is shown to be less performant than PPO, but does guarantee safety during training and deployment. A2C and DQN are typically able to produce high-performing policies, but are shown to have high variance between training runs and across hyperparameters. Therefore, PPO and MCTS-Train are the recommended



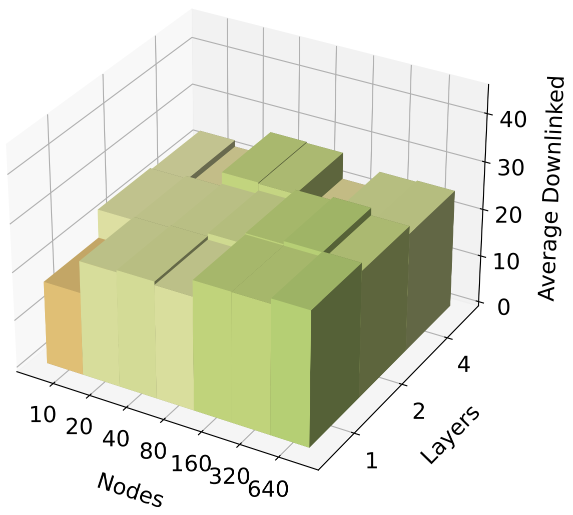
(a) Average reward.



(b) Success rate.



(c) Average number of imaged targets.



(d) Average number of downlinked targets.

Figure 5.13: MCTS-Train policy metrics. Reward components: $A = 0.9$, $B = 0.1$, $F = 0.25$.

algorithms for EOS scheduling, depending on the problem formulation.

This chapter also investigates long-duration deployment of the policies trained by MCTS-Train and PPO. Various methodologies are used to determine how policies can be used for planning horizons with 240 decision-making intervals for the SSAEO scheduling problem. The MCTS-Train policies trained using 45 decision-making intervals do not generalize to longer decision-making intervals without being wrapped with the safety shield. The PPO policies trained using 45 decision-making intervals perform slightly better, but to achieve comparable performance to shielded MCTS-Train, PPO must be trained with 240 decision-making intervals. Fortunately, this is a possibility for PPO, but unfortunately is not a possibility for MCTS-Train without a dramatic increase in computational resources. Regardless, MCTS-Train emerges as the recommended algorithm for the SSAEO scheduling problem due to its performance and safety guarantees in long duration deployment after being wrapped in the safety shield.

Finally, this chapter investigates how MCTS-Train and PPO respond to changes in the reward function. Different imaging, downlink, and failure constants are tested for each algorithm. PPO is shown to respond well to reward functions with more weight given to imaging, which results in slightly higher priority targets getting selected on average. Interestingly enough, the failure penalty is not shown to have a large impact on the performance of PPO. Because the episodes terminate when resource limits are exceeded, the decision-making agents are penalized by not being able to get more positive reward from collecting science data. For MCTS-Train, two separate failure penalties are tested. The higher failure penalty results in better policies that downlink more images on average. The larger failure penalty incentivizes the agent to downlink more in order to avoid data buffer overflows. Similar to PPO, however, the reduced failure penalty does not seem to impact the success rate of the decision-making agents. Again, agents are incentivized not to fail because they cannot collect more science data if they do.

Chapter 6

Multi-Satellite Agile EO Scheduling Problem

6.1 Introduction

The problem of scheduling the sequence of observation, downlink, and resource management tasks performed by a constellation of Earth-orbiting satellites with three-axis attitude control capabilities is commonly referred to as the multi-satellite agile Earth-observing (MSAEO) scheduling problem. Like the SSAEO scheduling problem, the primary challenge associated with the MSAEO scheduling problem is formulating performant, accurate, and computationally tractable problems that are flexible and fast enough to modify and re-solve in the inevitable event that replanning is required. However, the solutions should also be robust and scalable to support the addition and removal of satellites, which is a normal part of sustained constellation operations as new satellites are launched and old satellites are decommissioned. In the case of performance, problem formulations and solutions must optimal with respect to observations collected and downlinked while respecting the relevant resource constraints. Problem formulations must also be accurate, i.e. represent the real life problem with sufficient fidelity to minimize the frequency in which re-planning is required due to mismodeling. Finally, problem formulations and corresponding algorithms must be computationally tractable enough to solve during nominal operations and also fast enough to re-solve when replanning is required, such as when opportunistic science events present themselves. The same challenges in applying optimization-based solutions (i.e. MILP or GAs) to the SSAEO scheduling problem are present in the MSAEO scheduling problem. However, the computational cost of these algorithms is exacerbated by the increased number of decision-making agents and subsequent problem complexity.

While RL for single satellite EOS scheduling is gaining traction in the literature, few authors have explored utilizing RL for MSAEO scheduling. Cui et al. apply double Deep Q-Networks for communication scheduling of a constellation of Earth-orbiting satellites [74] and demonstrate that their algorithm is superior to a genetic algorithm in terms of performance and computation time. Dalin et al. formulate a scheduling problem for multi-satellite tasking and apply the multi-agent deep deterministic policy gradient (MADDPG) algorithm to solve the problem [75]. The performance of the MADDPG algorithm is shown to be comparable to other solvers for the problem. While each of these authors make important contributions to RL for MSAEO scheduling, the handling of resource constraints and their relationship to spacecraft position, velocity, and attitude is quite limited. The authors do not fully leverage the black-box optimization capabilities of reinforcement learning and rely on simple models of the problem. Furthermore, References [74] and [75] do not demonstrate the scalability of their algorithms to constellation parameters beyond those fixed during training.

Largely unexplored for MSAEO scheduling, multi-agent reinforcement learning (MARL) is a collection of problem formulations and algorithms associated with solving multi-agent decision-making problems. Various reward structures may be utilized, such as fully cooperative (agents receive a single global reward signal), competitive (each agent receives an individual reward signal), and mixed cooperative/competitive (agents receive local rewards, but their global reward is evaluated) [76]. While the fully cooperative reward structure intuitively makes sense, the challenge of credit assignment makes learning difficult. In addition to different reward structures, different training methodologies exist, such as independent learning (each agent independently learns its own policy, considering other agents part of the environment), centralized learning (a centralized policy is learned for the joint action space), and centralized training, decentralized execution (agents learn in a centralized manner, but are deployed in a decentralized manner). Independent learning algorithms include independent Q-learning (IQL) [77], independent advantage actor-critic (IA2C) [78], and independent proximal policy optimization (IPPO) [79]. Independent learning MARL paradigms must overcome the challenge of non-stationarity in the environment, as well as the challenge of the partial observability regarding other agents and their actions. However, in practice they work quite well. Centralized learning paradigms do not suffer from these issues, but the associated problems are difficult to solve due to the exponential increase in the joint action space with the number of decision-making agents. Any of the single agent RL algorithms discussed thus far can be utilized for the centralized training methodology if they are deployed over the full state and action space. However, they also cannot be executed in a decentralized manner. Centralized training, decentralized execution (CDTE) can overcome the limitations of each of the aforementioned training methodologies by leveraging the full observability of all agents during training, but only require that agents observe their local states during deployment. CDTE algorithms include multi-agent deep deterministic policy gradient (MADDPG) [80], counterfactual multi-agent (COMA) policy gradient [81], central-V [78], value decomposition networks (VDN) [82], and QMIX [83]. The relative performance of each independent learning and CDTE algorithm is problem dependent.

While RL may pose many benefits for MSAEO scheduling, the primary challenge is the computational complexity of the multi-agent problem, especially if a high-fidelity simulation is utilized. The most general formulation of a multi-agent RL problem is a decentralized partially observable Markov decision process (Dec-POMDP). However, a Dec-POMDP is non-deterministic exponentially complete (NEXP-Complete) [84]. If free communication is assumed, a Dec-POMDP can be reduced to a multi-agent MDP [56, 85]. However, the size of joint action space in both Dec-POMDPs and MMDPs is exponential in the number of decision-making agents. Chapters 4 and 5 demonstrate that a single agent can be trained in several hours to several days with MCTS-Train. A multi-agent reinforcement learning problem with comparable simulation fidelity could take much longer to train because of the exponential increase in computational complexity. While it may be tempting to deploy a multi-agent version of the other DRL algorithms in Chapter 5, it is likely that the gap in performance for the SSAEO scheduling will only grow in the MSAEO scheduling problem. To avoid the increase in computational complexity, but maintain the performance of MCTS-Train, this work deploys the decision-making agents trained in the SSAEO environment with

MCTS-Train and deploys them on-board each spacecraft in a Walker-delta constellation. While this problem formulation is suboptimal in terms of global reward because the decision-making agents are competing for reward, the size of the constellation may be readily changed without requiring retraining. To address this issue, this chapter also investigates higher level coordination for the target distribution so the agents are not competing for reward, but instead working through their individual local target lists while managing satellite resources such as power, on-board data storage, and reaction wheel speeds.

This chapter formulates a multi-satellite agile Earth-observing scheduling problem where a constellation of spacecraft in a Walker-delta formulation attempt to maximize the weighted sum of targets imaged and downlinked while avoiding resource constraint violations concerning power, on-board data storage, and reaction wheel speeds. An MDP formulation of the problem is created, and the various communication methods implemented in the multi-satellite scenario are described. The two methods of target distribution (i.e. first come, first served and the mixed integer programming techniques) are both described. Finally, the performance of the trained agents is benchmarked for each communication method, each target distribution method, and various Walker-delta constellation designs. This performance is compared to that of a genetic algorithm as well, which can provide a benchmark of performance for a centralized algorithm. Part of this work is published in References [86] and [87].

6.2 Problem Formulation

6.2.1 MDP Formulation

The multi-satellite agile Earth-observing scheduling problem extends the SSAEO scheduling problem to multiple satellites in a Walker-delta constellation. N satellites are distributed evenly among P orbit planes. The orbital planes are distributed at $360/P$ deg intervals of the longitude of ascending node. A phasing factor between the planes may also be specified, which determines the offset in true anomaly between satellites in adjacent planes. In this work, a phasing factor of 0 is

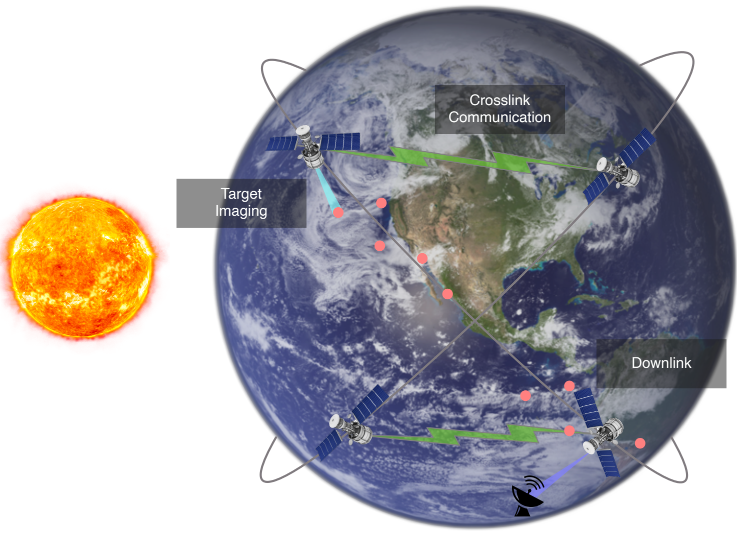


Figure 6.1: Multi-satellite agile Earth-observing scheduling problem. A constellation of satellites attempts to maximize the local weighted sum of imaging targets imaged and downlinked.

utilized. The satellites in the constellation have access to a global set of targets, M. Furthermore, each satellite k has its own set of targets, T_k, but satellites may share targets within M. In this chapter, the decision-making agents attempt to maximize local reward. That is, each satellite attempts to maximize the weighted sum of targets collected and downlinked within its local target set T_k. The satellites update their local target lists through communication with the other satellites. This concept is demonstrated in Figure 6.1.

Each satellite in the Walker-delta constellation maintains an observation over its local state, $s_i^k \in \mathcal{S}^k$. The full state is now $s_i = \{s_i^1, \cdots, s_i^k\}$, and the state space is $S = \mathcal{S}^1 \times \cdots \times \mathcal{S}^k$. The state evolves based on the underlying system dynamics and actions taken by each satellite. The action space is now a joint action space represented as $\mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^k$. Each decision-making agent takes actions following its local copy of the policy, $a_i^k = \pi^k(s_i^k)$. The generative transition function and the reward function are both now a function of the complete state and joint actions. The generative transition function is given by $s_{i+1}, r_i^1, \cdots, r_i^k \sim G(s_i, \mathbf{a}_i)$, and the joint reward function is given by $\mathbf{R}(s, \mathbf{a}) = (R^1(s, \mathbf{a}), \cdots, R^k(s, \mathbf{a}))$. The local reward function of each agent is the same as it is for the SSAEO scheduling problem, but the environment now enumerates through M instead of T_k to check if a target was imaged or downlinked for the first time or not. If another satellite has already imaged or downlinked a target, no new reward is returned. The MDP formulation of the MSAEO is technically a Markov Game [56]. However, the agents on-board each satellite do not evolve their strategies based on the actions of the other agents, so the other agents are considered part of the environment. From a single agent's perspective, the two changes from the SSAEO scheduling problem to the MSAEO scheduling problem are A.) the potential that a target in the action space (or more precisely the set U) is imaged at the same decision interval, and B.) changing distributions of the local target set T_k, which agent k cannot observe past U anyway. The frequency of occurrence of the former change is the only change that may necessitate modifying the single agent training paradigm used in this chapter. This is discussed in the results.

6.2.2 Simulation Architecture

The generative transition function, $s_{i+1}, r_i^1, \cdots, r_i^k \sim G(s_i, \mathbf{a}_i)$, of the MSAEO scheduling problem is also implemented using a Gymnasium environment wrapping a Basilisk simulation. The components of the simulation architecture are identical for both the MSAEO and SSAEO scheduling problems, and a summary of the simulation is provided in Figure 6.2. The Basilisk simulation includes three separate classes for 1.) environment modules, 2.) dynamics modules, and 3.) FSW modules. A single environment class is instantiated for the entire simulation, but dynamics and FSW classes are instantiated for each satellite. The architecture allows for the number of spacecraft to easily scale up and down using only a few lines of code.

The environment class contains modules for gravity (the Earth and sun), eclipse, each of the seven ground stations available for downlink, imaging targets, an atmospheric density model, and a random disturbance torque that helps build up momentum in the reaction wheels. The dynamics class of each satellite contains modules for the power system (i.e. batteries, solar panels, instrument and transmitter power sinks, etc.), data management system (i.e. data buffer, instrument, and transmitter), and attitude control system (reaction wheels and thrusters). Furthermore, spacecraft location modules are instantiated for each satellite and connected to the other satellites to determine when line-of-sight access occurs. Finally, the flight software class of each satellite contains

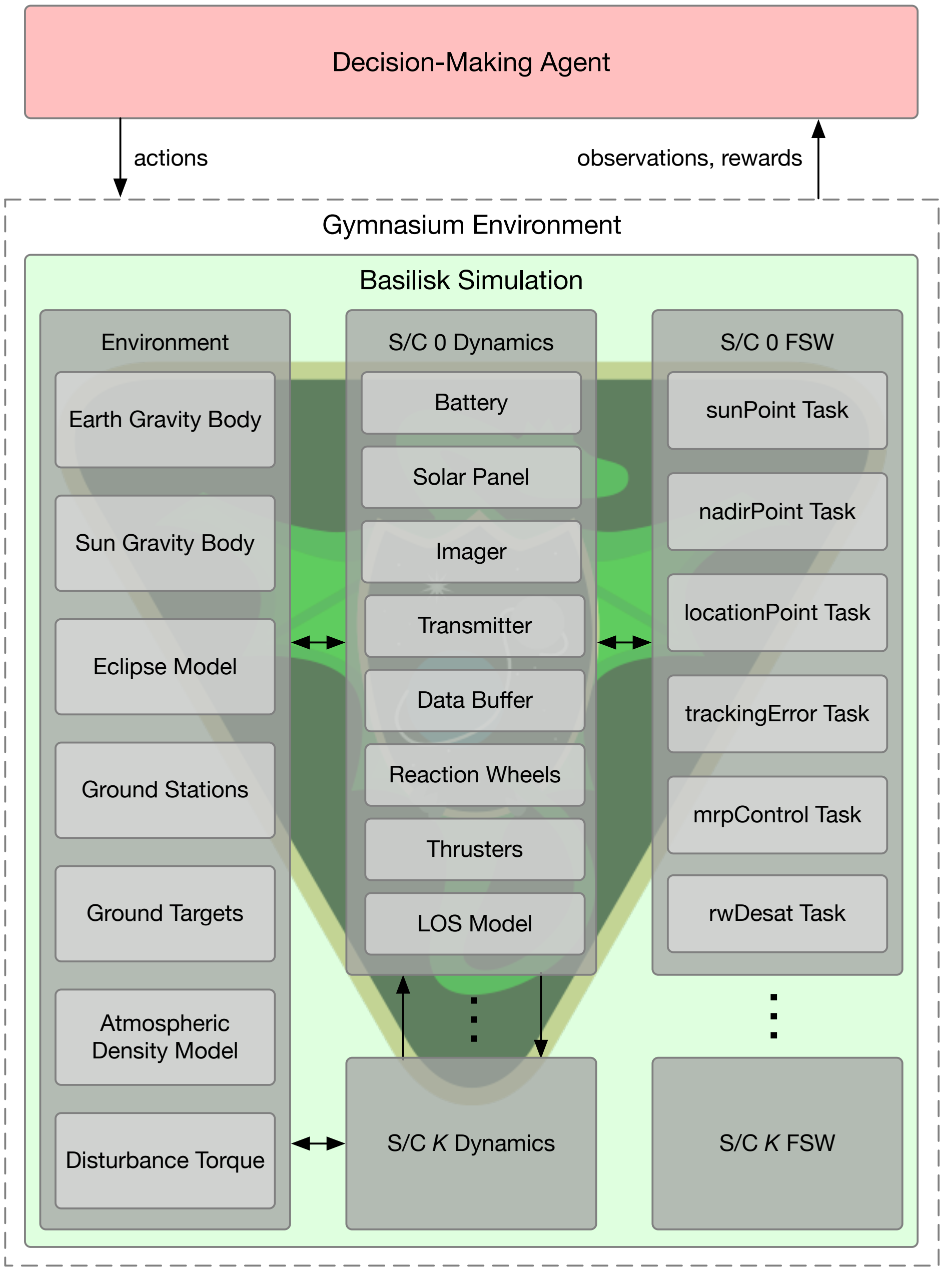
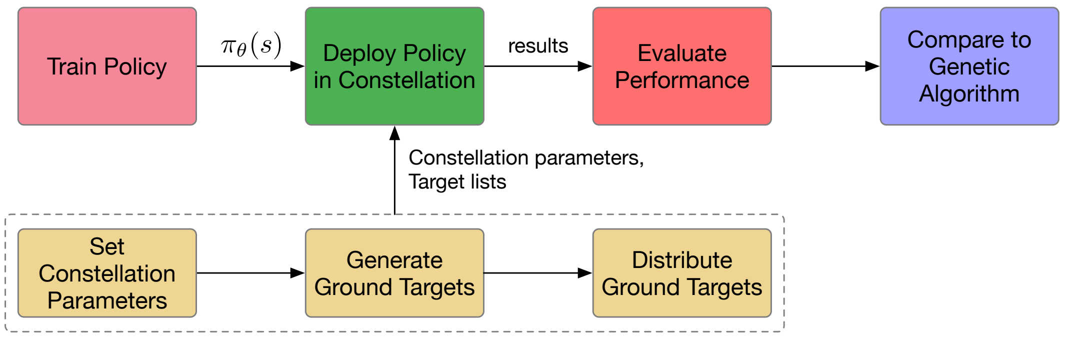


Figure 6.2: Multi-satellite Basilisk simulation architecture.

numerous tasks. The `sunPoint`, `nadirPoint`, `locationPoint`, and `trackingError` task provide an attitude reference to the `mrpControl` task, which uses a MRP-based feedback control law to compute reaction wheel torques. Finally, the `rwDesat` task contains the modules that map reaction wheel momentum to thruster commands. The source code for the MSAEO scheduling problem may



Train Policy
$\pi_\theta(s)$
Deploy Policy in Constellation
results
Evaluate Performance
Compare to Genetic Algorithm
Constellation parameters, Target lists
Set Constellation Parameters
Generate Ground Targets
Distribute Ground Targets

Figure 6.3: Policy deployment pipeline.

be found on the develop branch of the bsk_rl library[1] under the name and `multiSAtAgileEOS`. The satellite and simulation parameters are provided in Chapter 3. However, like in Chapter 5, the longitude of ascending node is sampled from $\mathcal{U}[0, 360)$ deg.

6.3 Methods

This section provides an overview of how the decision-making agents are trained and deployed in the constellation. In the first step, the decision-making agents are trained in the single satellite environment. The trained policy is wrapped within a safety shield that prevents the decision-making agents from taking unsafe actions. Then, the policy is deployed on each satellite in a constellation defined using a set of Walker-delta parameters, imaging targets, and communication assumptions. After the policy is deployed in the environment, the performance subject to the Walker-delta parameters, distributed target set, and selected communication assumption is evaluated. This performance is then compared to the performance of a genetic algorithm for a subset of the Walker-delta parameters explored. The deployment pipeline is summarized in Figure 6.3. This section details the policy training, policy deployment, and ground target generation blocks of the pipeline. The communication methods, which are contained within the constellation parameterization block, and the genetic algorithm used for comparison purposes are also described.

[1] https://github.com/AVSLab/bsk_rl

6.3.1 Single Agent Training

The single agent training process is described in detail in Chapters 4 and 5, but will be summarized here. MCTS-Train is used to generate a neural network approximation of the state-action value function, $Q_\theta(s, a)$. MCTS-Train uses Monte Carlo tree search (MCTS), an online tree search algorithm, to find optimal solutions to the planning problem. At every step through the environment, MCTS runs a number of simulations to compute an estimate of the state-action value function. After all the simulations have been completed, the action that maximizes the state-action value function is selected. The environment transitions to the next state, and the process repeats until the end of the planning horizon. After the end of the planning horizon is reached, the state-action value estimates along the main trajectory of the search tree are collected and added to a data set. This data set is then regressed over with a feedforward neural network to compute a neural network approximation of the state-action value function, $Q_\theta(s, a)$. The parameterized state action value function is used to create a parameterized policy, $\pi_\theta(s)$, using Equation 4.9.

6.3.2 Multi-Agent Deployment

The decision-making agents, or policies, are trained using the MCTS-Train pipeline. After training, the policies are deployed on-board each satellite in the Walker-delta constellation. The policy on-board each satellite is wrapped with a safety shield that ensures the decision-making agent only takes safe actions. For instance, if decision-making agent attempts to take an image when the data buffer is one image away from overfilling, the safety shield will override the decision with a safe action (i.e. downlink). A visual representation of the shield in action is provided in Figure 2.4. Both the decision-making agent and the safety shield receive an observation from the environment. The decision-making agent passes an action to the safety shield, which evaluates the observation and action to ensure a safe action is passed to the environment. The details regarding the safety shield are provided in Chapters 2.3.6 and 4.2.2.2.

6.3.3 Communication Methods

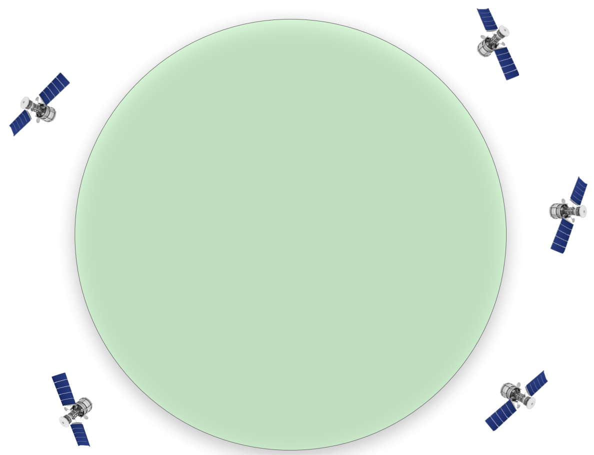
Communication between the satellites is used to locally update which targets have been imaged and downlinked to help ensure that the duplication of efforts is minimal. At the end of each decision-making interval, the satellites use the selected communication method to update their local lists of targets, T_k. Four separate communication methods are implemented: no communication, single degree line-of-sight communication, multi-degree line-of-sight communication, and free communication. These are displayed in Figure 6.4.

6.3.3.1 No Communication

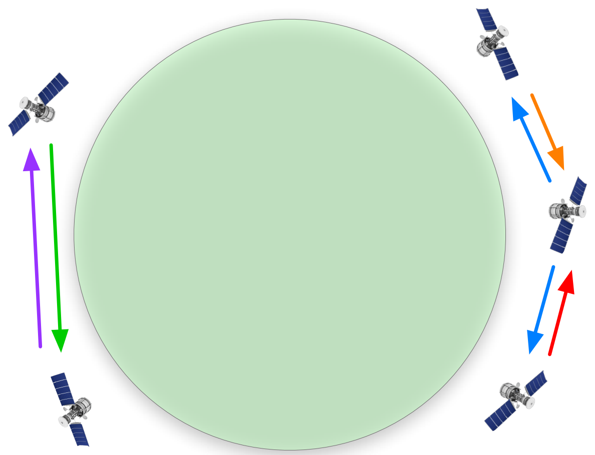
The no communication model assumes that the satellites do not update their local target lists. The local target sets T_k are never updated to include which targets have been imaged or downlinked by other satellites in the constellation.

6.3.3.2 Single Degree Line-of-Sight Communication

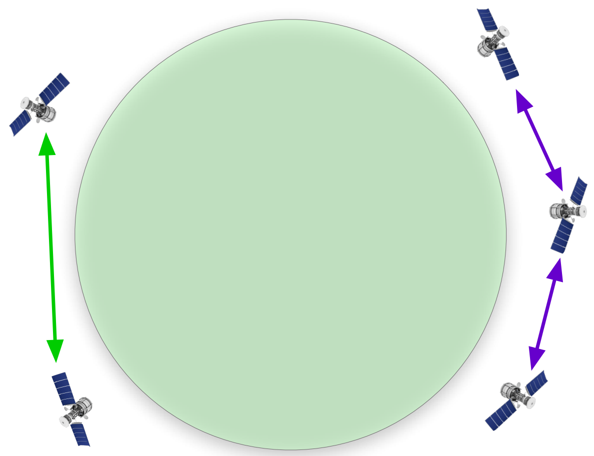
The single degree line-of-sight communication assumption is meant to represent a constellation with limited crosslink communication capabilities. Line-of-sight connectivity between the satellites is defined as a straight-line connecting two satellites that does not intersect the Earth plus 100 km of atmosphere above the surface of the Earth. Each satellite updates its local list of imaging targets with the neighbors it is directly connected to using only one iteration of communication. Imagine a scenario in which spacecraft A has line-of-sight communication with spacecraft B, but spacecraft B has line-of-sight communication with both spacecraft A and spacecraft C. Spacecraft A will not receive information about which targets spacecraft C has imaged and downlinked; it will only receive information about which targets spacecraft B has imaged and downlinked. This is demonstrated in Figure 6.4b.



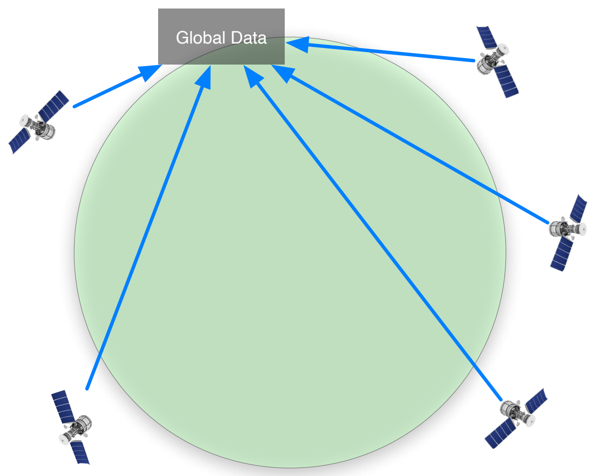
(a) No communication.



(b) Single degree line-of-sight communication.



(c) Multi-degree line-of-sight communication.



(d) Free communication.

Figure 6.4: Communication methods.

6.3.3.3 Multi-Degree Line-of-Sight Communication

The multi-degree line-of-sight communication assumption is meant to represent a constellation with near unlimited crosslink communication bandwidth. If the previous example is used again, spacecraft A will now receive information about which targets both spacecraft B and C have imaged and downlinked.

6.3.3.4 Free Communication

The free communication case is meant to represent a constellation with near constant access to communication resources, either ground- or space-based. This could include a large network of ground stations or a dedicated constellation for communication routing. In the free communication assumption, every satellite has access to which targets have been imaged and downlinked in the global target set M, and their local target lists are updated accordingly.

6.3.4 Ground Target Distribution

The set of M imaging targets is generated using uniformly distributed unit vectors projected onto the surface of the Earth. This chapter investigates two different methods for distributing the imaging targets between the satellites in the constellation, one centralized and one decentralized. The first method distributes the imaging targets solely based on access time in a first come, first serve manner where imaging targets may be shared between satellites. The second method optimally distributes the targets using a mixed integer program, and no imaging targets are shared between satellites.

6.3.4.1 Ordered Access Target Distribution

The first method of target distribution creates a set of local targets for each satellite ordered by the access time to that target. Targets may be shared between satellites using this method. The algorithm for this target distribution method is provided in Algorithm 9. Initial conditions are first generated for each satellite. Then, each satellite is looped through to create the list of local targets. The local target set is initialized, the spacecraft trajectory is propagated for the duration of the planning horizon, and the access times for each target are computed. Then, for each interval and each target, if the satellite has access to the target, the target is added to the local list.

Algorithm 9 Ordered access target distribution.

```
1: initialize set of initial conditions for $K$ spacecraft
2: for spacecraft k = 1:K
3:     initialize local target set $T_k$
4:     propagate spacecraft trajectory
5:     pull access times $o_{i,j,k}$
6:     for $i \in I$
7:         for $m \in M$
8:             if $o_{i,m,k}$
9:                 $\boldsymbol{T}_k \cup \{m\}$
10: assign local target sets to spacecraft initial conditions
```

6.3.4.2 Mixed Integer Programming Target Distribution

The second method of target distribution utilizes a mixed integer program to generate an optimal distribution of targets. The purpose of using this method is to determine the impact that centralized coordination has on performance. The objective of the integer program is to maximize the weighted sum of targets distributed between the satellites. This objective function is provided in Equation 6.1, where I is the set of decision intervals, M is the set of global targets, and K is the set of satellites. $x_{i,m,k} \in \{0, 1\}$ is the binary decision variable for whether or not a target $m \in M$ is assigned to spacecraft $k \in K$ at interval $i \in I$, and $p_m \in \mathbb{R}^+$ is the priority of target m. $o_{i,m,k} \in \{0, 1\}$ is a binary variable representing the access of spacecraft k to target m at interval i. The integer program includes several constraints. The first constraint, provided in Equation 6.2, ensures that a ground target is collected no more than one time over the planning horizon. The second constraint, provided in Equation 6.3, ensures that each satellite collects at most one target at every decision interval. Finally, the last constraint, Equation 6.4, ensures that imaging targets are only collected when access is available.

$$\max \sum_{i \in I} \sum_{m \in M} \sum_{k \in K} \frac{x_{i,m,k}}{p_m} \tag{6.1}$$

Algorithm 10 Mixed integer programming target distribution.

```
1: initialize set of initial conditions for $K$ spacecraft
2: for spacecraft k = 1:K
3:     propagate spacecraft trajectory
4:     pull access times $o_{i,j,k}$
5: construct MIP formulation
6: solve optimization problem
7: construct local target sets $T_k$
8: assign local target sets to spacecraft initial conditions
```

s.t.

$$\sum_{i \in I} \sum_{k \in K} x_{i,m,k} \leq 1 \; \forall \; m \in M \tag{6.2}$$

$$\sum_{m \in M} x_{i,m,k} \leq 1 \; \forall \; i \in I, \; k \in K \tag{6.3}$$

$$x_{i,m,k} \leq o_{i,m,k} \; \forall \; i \in I, \; m \in M, \; k \in K \tag{6.4}$$

Note that the target distribution program does not account for data buffer, reaction wheel speed, and power constraints. While data buffer and power constraints are straightforward to model with an integer program if linearity assumptions are made regarding their dynamics, reaction wheel speeds are not because of the highly non-linear dynamics involved. With this target distribution method, the trained decision-making agents are in charge of resource management, and the mixed integer program is in charge of supplying the decision-making agent with the next best target to image.

The algorithm for the MIP target distribution method is provided in Algorithm 10. Similar to the ordered access distribution, the set of initial conditions is first generated. Then, the spacecraft trajectory is propagated and the access times are computed. The mixed integer program is then implemented using the Python-MIP[2] optimization package and solved using the default branch & cut algorithm. After an optimal solution is generated, the target sets are assigned to each satellite.

[2] https://www.python-mip.com/

6.3.5 Genetic Algorithm

A genetic algorithm is implemented to compare the solution method presented in this chapter to a method that optimizes over the entire action space, $\mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^k$, of the MSAEO scheduling problem. For the MSAEO scheduling problem, each sequence of actions is simply input into the MSAEO simulator, and the reward function in Equation 3.7 is evaluated for each agent and summed. The pseudocode for the genetic algorithm is provided in Algorithm 8.

6.4 Results

6.4.1 Communication Methods

To determine how assumptions regarding communication between satellites impact performance, several benchmark experiments are performed for each communication method deployed in different Walker-delta constellation designs. In the first experiment, each communication method is tested in a constellation of K satellites that reside in a single orbital plane. In the second experiment, each communication method is tested in a constellation of satellites distributed among P orbital planes.

6.4.1.1 Single Plane Results

In the single plane experiments, a constellation of $K = \{4, 7, 10, 15, 20, 30, 40\}$ satellites are deployed in a single plane at 45 degrees inclination. The purpose of this experiment is to determine how performance of the trained agent depends on intra-plane communication. Analytical predictions about this performance can be readily made based solely on whether or not the satellites in the orbital plane can communicate with one another or not. First and foremost, it is always expected that the free communication assumption will outperform the no communication assumption. The reason for this is that the decision-making agents will never be aware if another satellite has already imaged or downlinked a potential target. However, the performance of the line-of-sight communication assumption is not as easily predicted. There will be some critical number of satellites, K^*, that determines whether or not the Earth occludes communication. For $K < K^*$, there will never be line-of-sight communication between the satellites, and the performance of the line-of-sight communication assumptions should match that of no communication. For $K > K^*$, the satellites will always maintain line-of-sight communication with one another, and the performance of the line-of-sight communication assumption should match that of free communication.

Assuming a semi-major axis of 6871 km and occlusion occurring for altitudes less than 100 km (due to atmospheric interference), the critical number of satellites, K^*, may be computed as follows, where θ^* is the angle between the right triangle formed by the satellite's radius and the occluding radius:

$$K^* = \frac{\pi}{\theta^*} = \frac{\pi}{\cos^{-1}\left(\frac{R_E + 100\text{km}}{R_E + 500\text{km}}\right)} = 9.2 \text{ satellites} \tag{6.5}$$

A diagram for this is provided in Figure 6.5. Therefore, for 9 or fewer satellites, line-of-sight communication is identical to no communication. For 10 or more satellites, line-of-sight communication will approximate free communication.

An experiment is performed for the described Walker-delta constellations; 16 samples are generated for each combination of constellation design and communication model. The results of this experiment are provided in Figures 6.6, 6.7, and 6.8. In Figure 6.6, 2D and 3D views of the global and local reward are plotted. Global reward is the sum of reward across all satellites, and local reward is the average reward of each satellite following Equation 3.7. The first observation to note is that the analytical prediction regarding when line-of-sight communication approximates none or free communication matches the experimental results. For the blue and orange curves (4 and 7 satellites), line-of-sight communication approximately matches no communication. For the green, red, purple, brown, and pink curves ($\geq$ 10 satellites), line-of-sight communication approximately matches free communication. The second observation to note is that the performance difference between the single- and multi-degree line-of-sight communication assumptions is not discernible. The two communication methods perform approximately the same. This is due to the fact that one-way information sharing between neighbors (i.e. single degree line-of-sight communication) is

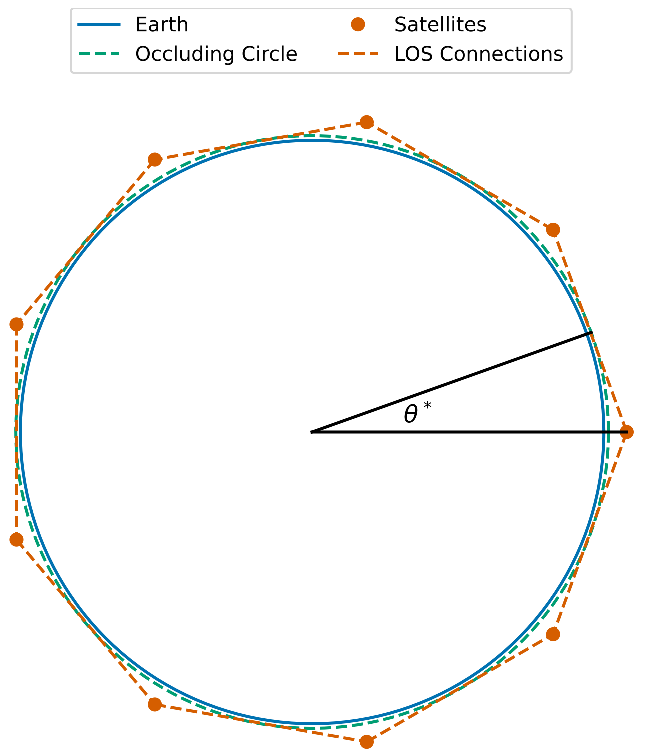
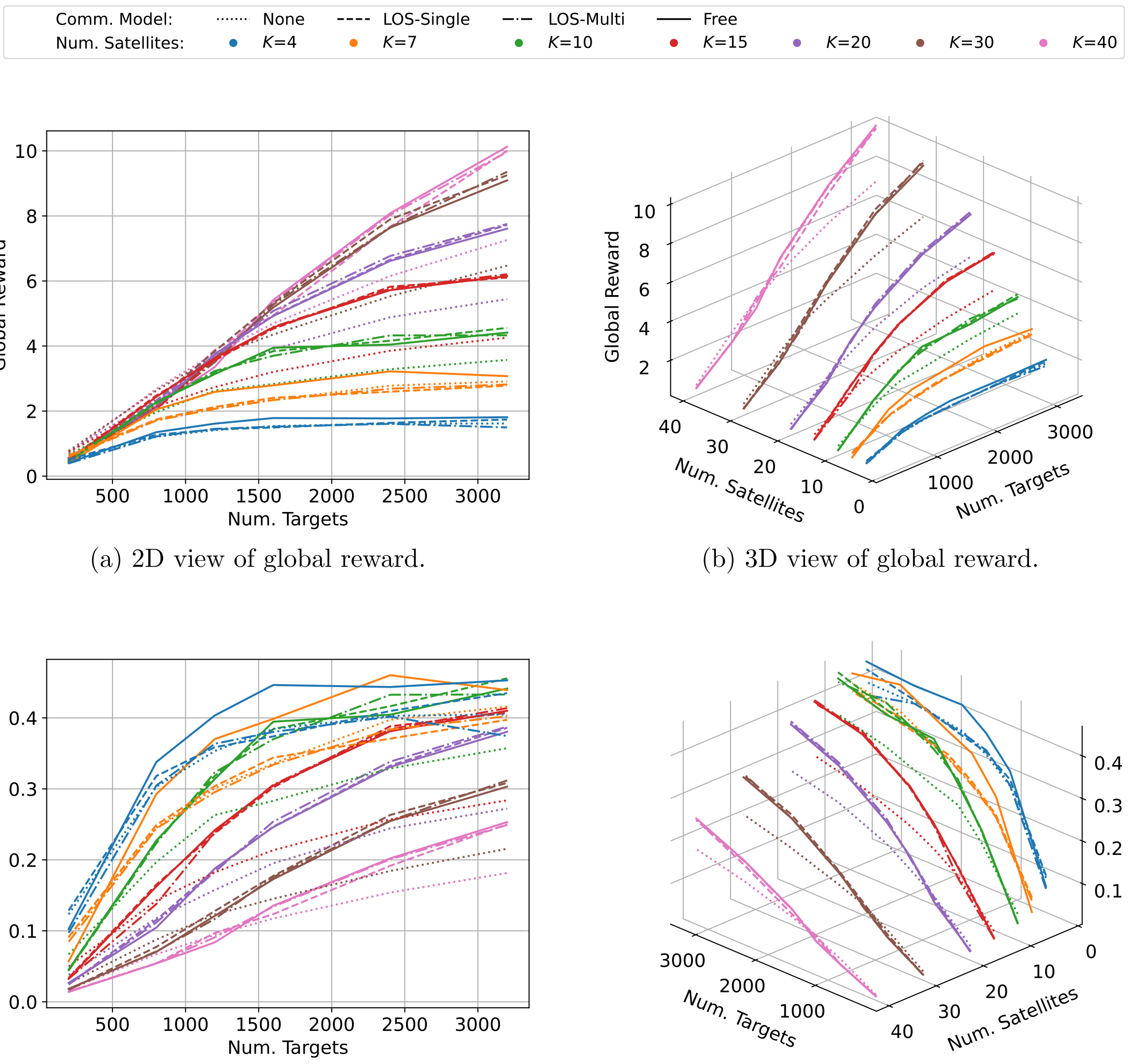


Figure 6.5: The critical angle, θ^*, which determines when LOS communication is not possible.

sufficient to ensure neighboring satellites do not duplicate one another satellite's efforts.

In addition to observations regarding the performance of each communication method, observations can be made about the dependency of global and local reward on the size of the constellation and the number of global targets. In general, more global targets correlates with higher reward. For smaller constellations, the satellites become saturated with imaging tasks, and the reward plateaus. Furthermore, more satellites generally results in higher global reward. This intuitively makes sense. As satellites are added to the constellation, there are more resources available to image and downlink targets. However, local reward decreases as more satellites are added to the constellation. Because there is more competition for the available imaging targets, and because many of the satellites share the same imaging targets, local reward decreases with an increase in satellites.

In Figure 6.7, the performance of the deployed agents is displayed in terms of the unique number of imaged and downlinked targets. Note that the shape of the unique number of imaged



(a) 2D view of global reward.

(b) 3D view of global reward.

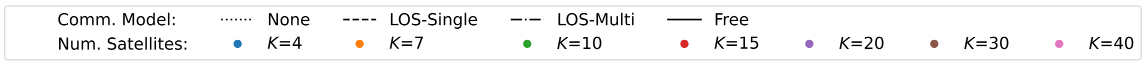
(c) 2D view of local reward.

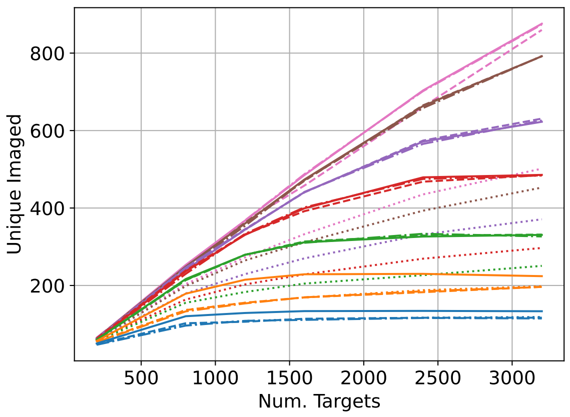
(d) 3D view of local reward.

Figure 6.6: Global and local reward for the single plane experiment.

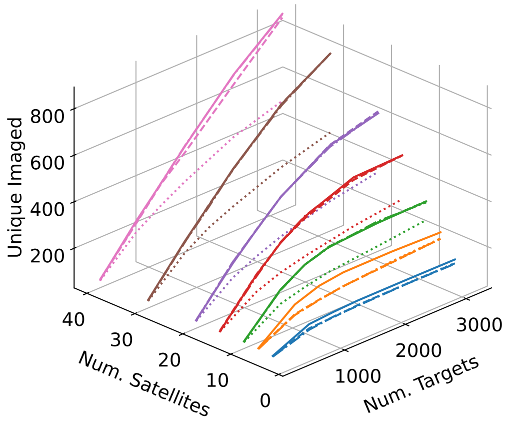
and downlinked targets match one another as well as the shape of the reward function.

In Figure 6.8, the percent of the imaged imaging targets that are unique is plotted. As expected, the no communication assumption results in a low percentage of images that are unique. For a large constellation of 40 satellites with only 200 imaging targets, less than 10% of the imaged targets are unique. For a small constellation of 4 satellites with 3200 imaging targets, about 90% of the imaged imaging targets are unique. The percentage of unique targets for the free communication

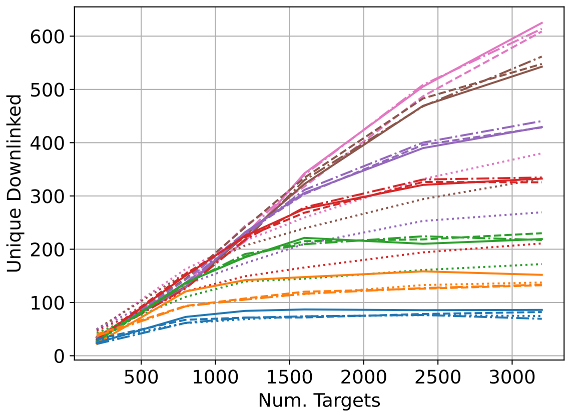




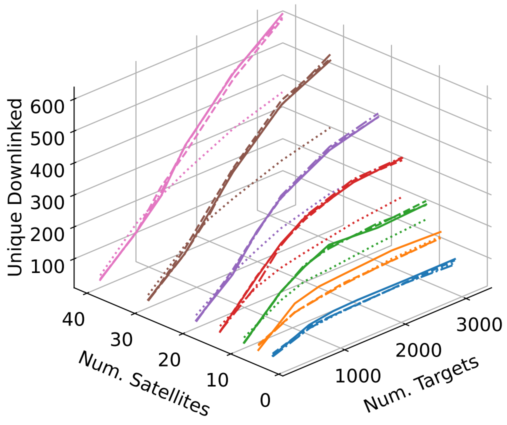
(a) 2D view of imaged targets.



(b) 3D view of imaged targets.



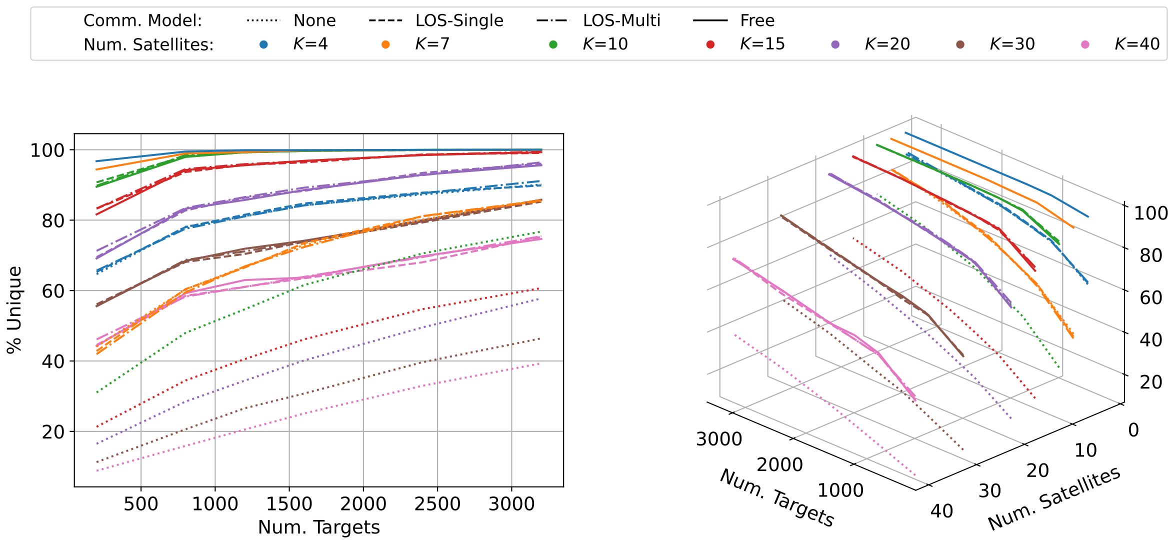
(c) 2D view of downlinked targets.



(d) 3D view of downlinked targets.

Figure 6.7: Global numbers of unique imaged and downlinked targets for the single plane experiment.

case is heavily dependent on the size of the constellation as well. Only about 50% - 80% of the targets imaged assuming free communication in a constellation of 40 satellites are unique. The reason for this is that the satellites are not coordinating during a given decision-making interval. It is possible, and depending on the priority of the target likely, that more than one satellite will attempt to image the same ground target at the same decision-making interval. This rate of target duplication indicates that modifications to the SSAEO training environment where targets



(a) 2D view.

(b) 3D view.

Figure 6.8: Percent of imaged targets that are unique for the single plane experiment.

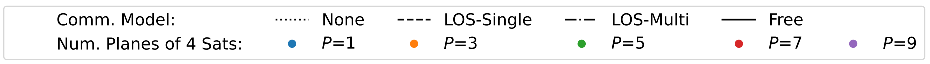
in the set U randomly disappear, to simulate the target being imaged by another satellite, may be beneficial.

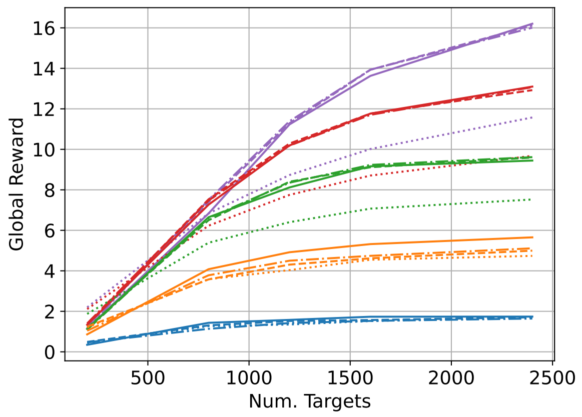
6.4.1.2 Multi-Plane Results

Duplicate benchmarks are performed for a Walker-delta constellation with multiple planes. A set of $P = \{1, 3, 5, 7, 9\}$ planes with four satellites in each plane at a 45 degree inclination is benchmarked for $M = \{200, 800, 1200, 1600, 2400\}$ global imaging targets. A phasing factor of 0 is utilized. Similar to the single plane experiments, there exists some number of planes P^* where two satellites in adjacent planes at the equator can communicate with one another. Using the same assumption as the single plane case, this number is computed as follows:

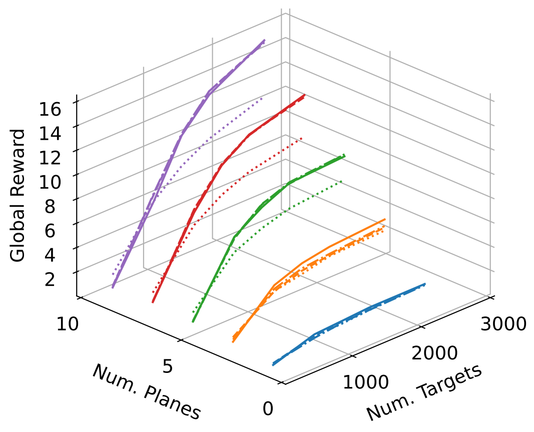
$$P^* = \frac{\pi}{2\cos^{-1}\left(\frac{R_E + 100\text{km}}{R_E + 500\text{km}}\right)} = \frac{1}{2}K^* = 4.6 \text{ planes} \tag{6.6}$$

Therefore, four or less planes will result in no communication at the equator. Five or more planes will result in communication at the equator. To determine how the number of planes impacts performance, four satellites per plane is selected such that there is no intra-plane communication.

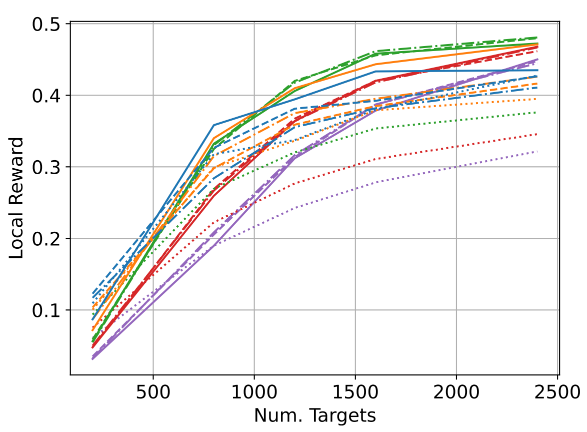




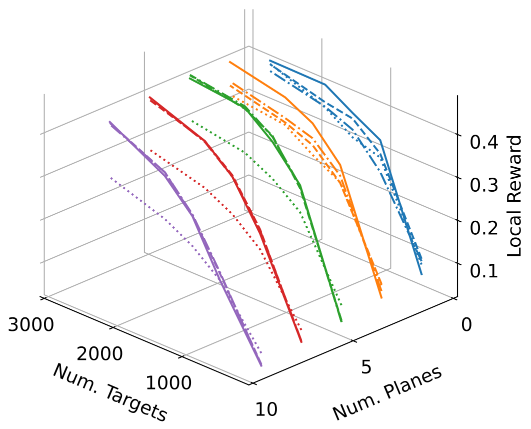
(a) 2D view of global reward.



(b) 3D view of global reward.



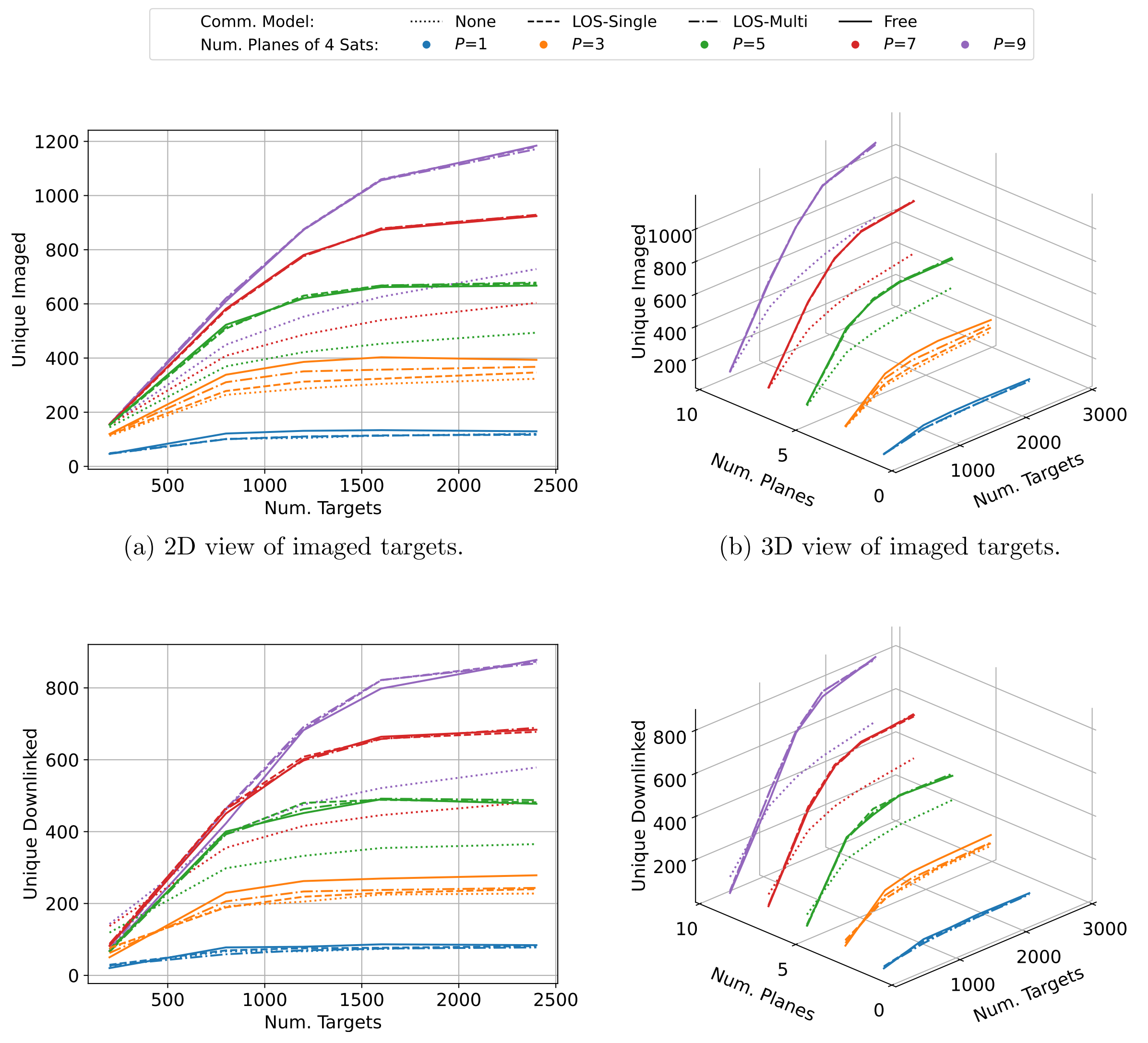
(c) 2D view of local reward.



(d) 3D view of local reward.

Figure 6.9: Global and local reward for the multi-plane experiment.

The global and local reward of the experiment is provided in Figure 6.9. The local reward plots in Figures 6.9c and 6.9d demonstrate the impact of the inter-plane communication on performance. For $P = \{1, 3\}$ planes, line-of-sight communication more closely matches that of no communication. However, because there is intermittent communication where orbit lines intersect, the line-of-sight communication assumption does not match the no communication assumption like it did for the single plane experiments.



(a) 2D view of imaged targets.

(b) 3D view of imaged targets.

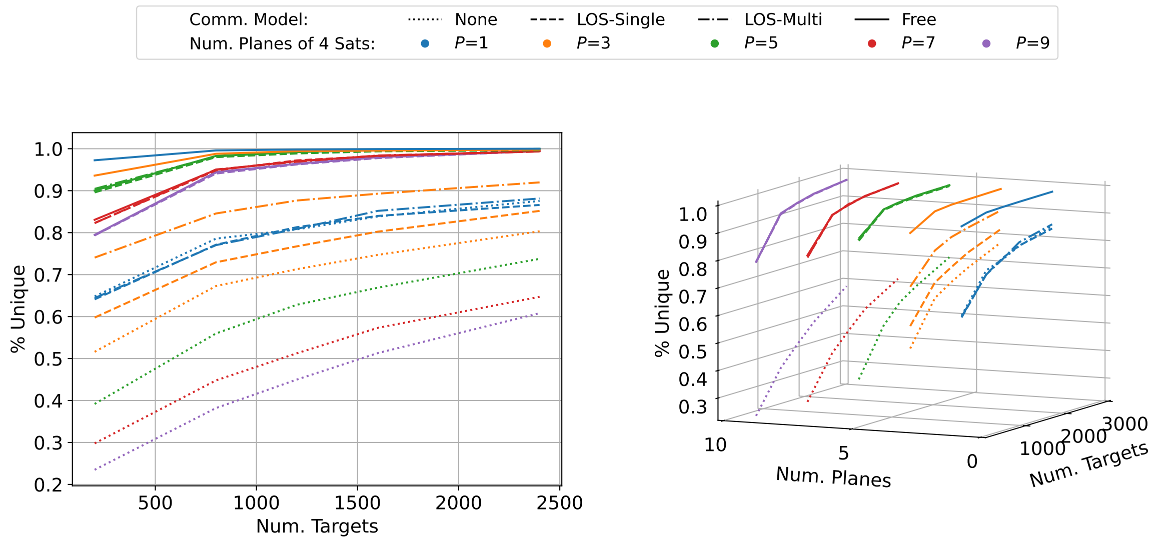
(c) 2D view of downlinked targets.

(d) 3D view of downlinked targets.

Figure 6.10: Global numbers of unique imaged and downlinked targets for the multi-plane experiment.

The number of unique imaged and downlinked targets are provided in Figure 6.10. As expected, the no communication assumption results in the fewest amount of uniquely imaged and downlinked targets. The free communication assumption results in the most amount of uniquely imaged and downlinked targets.

Finally, the percent of imaged targets that are unique are provided in Figure 6.11. These experimental results match that of the single plane experiment. No communication results in the



(a) 2D view. (b) 3D view.

Figure 6.11: Percent of imaged targets that are unique for the multi-plane experiment.

lowest percentage of targets that are unique. Furthermore, more planes and more satellites results in fewer uniquely imaged and downlinked targets because the probability that two satellites will image the same target at the same decision-making interval increases. This probability decreases as the number of global targets increases, but the trend is still present. Finally, it's worth noting that for the multi-plane Walker-delta formations, the percent of unique targets is higher than that of the single plane Walker-delta formations. This is because the satellites are more spread out in the multi-plane formations, which results in a lower probability of two satellites imaging the same target at the same decision-making interval. Modifications to the SSAEO training paradigm may still improve performance for multi-plane formations, but to a lesser degree.

6.4.1.3 Communication Method Discussion

Several insights may be drawn from the results of the communication experiments. First, it is evident that the communication assumption has a significant impact on the performance of the constellation. However, there is little to no difference between the free and line-of-sight communication assumptions if there are enough satellites in a single plane to communicate with one another. The more satellites available in a plane, the more targets that can be imaged and downlinked. However, the probability that two satellites will image the same target increases as well, and the overall efficiency of the constellation is decreased, as evidenced by Figure 6.8. If there are enough satellites in the constellation such that LOS communication approximates free communication, then there will be an overlap in the targets available to neighboring satellites (and even non-neighboring satellites if the plane is dense enough). The proposed decentralized decision-making architecture cannot ensure that the satellites do not image the same targets as the individual decision-making agents do not communicate intent to one another, but only communicate the targets they have already imaged and downlinked. Therefore, the satellites will not know what targets their neighbors are planning to image. This is a significant limitation of the proposed decentralized decision-making architecture. However, the results of the communication experiments show that this limitation is not a significant issue if the constellation is small enough. In that case, performance is limited by whether or not the satellites can communicate at all.

Another insight deals with the number of planes. When an equal number of satellites are distributed among planes such that no intra-plane communication is available, but inter-plane communication is available, the performance of the constellation improves significantly. This is because the satellites within the plane do not have to worry about imaging the same targets as their neighbors and benefit from information being propagated between the planes. When comparing the efficiency metrics from the single- and multi-plane results (Figures 6.8 and 6.11), it is evident that the multi-plane results are significantly better. In this case, a coordinated approach would likely have less of a potential to improve the results, unless the use of coordination and intent communication can help improve resource management for a single satellite, which can better prepare it for future science opportunities.

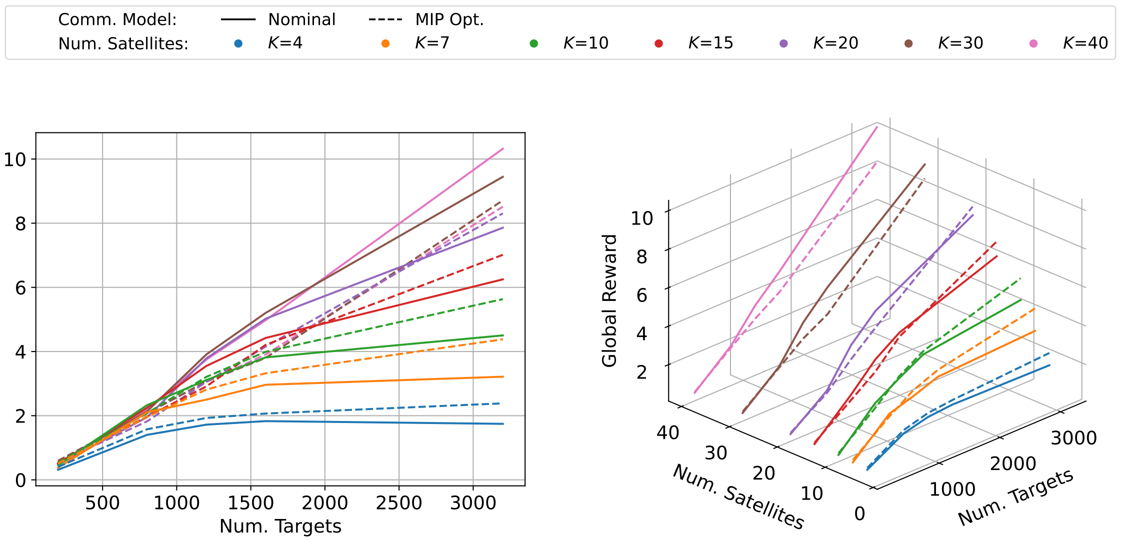
Lastly, while the MSAEO scheduling solution provided in this chapter is posed for deployment on board real satellites, the utility of the proposed method for evaluating constellation designs is quite evident. The amount of science collected and the overall efficiency of a particular constellation design can be evaluated relatively quickly. A grid search over the constellation parameters can be performed to maximize or minimize whatever metric the constellation designer is interested in.

6.4.2 Ground Target Distribution Comparison

The last set of experiments addresses the question of how best to distribute targets. Two approaches are taken - one centralized and one decentralized. The centralized approach formulates a MIP optimization problem to distribute the targets between satellites such that no targets are shared. The decentralized approach assigns imaging targets to satellites as they are available, and the first satellite to capture and downlink the ground target receives the reward. For each comparison, the centralized approach utilizes a neural network trained with a $|U_k| = 1$. The decision-making agent only looks one target ahead. The decentralized approach utilizes a neural network trained with a $|U_k| = 3$. The decision-making agent looks three targets ahead. The reason for this is that the centralized distribution approach effectively prunes the list of targets to provide the next best one to the decision-making agent. Therefore, the look-ahead capabilities provided by $|U_k| = 3$ is not necessary for the centralized approach. Furthermore, the decision-making agents from Chapter 5 are trained assuming a much larger target density than what the centralized approach will output, which may negatively impact policies trained for $|U_k| = 3$.

6.4.2.1 Single Plane Results

The first set of ground target distribution experiments compares each method for varying numbers of satellites in a single plane. The parameters of the experiment are the exact same as the parameters of the communication experiments. For the decentralized distribution approach, free communication between satellites is assumed. For the centralized distribution, no communication between satellites is necessary. For each initial condition, provided that the access intervals are already computed, it takes on average 0.73 seconds to solve the MIP for the smallest case of satellites (K=4) and imaging targets (200). However, it takes an average of 170 seconds to solve for the largest case of satellites (K=40) and imaging targets (3200). The global reward for each method is provided in Figure 6.12. A general trend can be observed with these plots. For large



(a) 2D view.

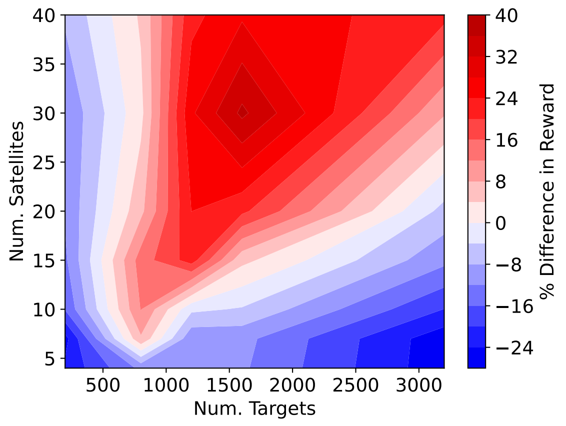
(b) 3D view.

Figure 6.12: A comparison of global reward for nominal vs. MIP target distribution methods.

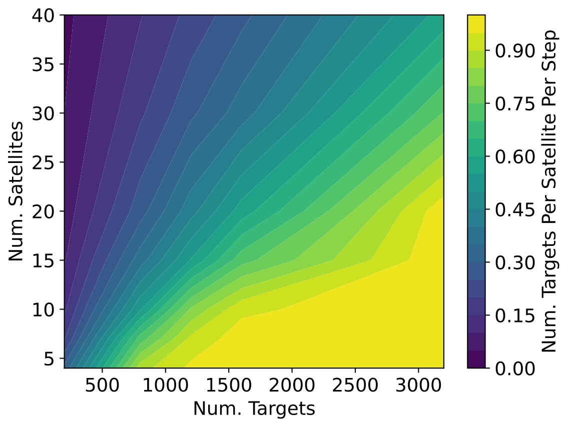
constellations ($\geq$ 30 satellites), the decentralized target distribution approach performs best for all numbers of global targets. For small constellations ($\leq$ 10) satellites, the centralized distribution method performs better for almost all numbers of imaging targets. For constellations of 15-20 satellites, the centralized approach performs best for large numbers of targets, but worse for small numbers of targets.

In Figure 6.13, more specific metrics are plotted to better understand where and why each method performs best. In Figure 6.13a, the percent difference in reward between the decentralized and centralized methods is plotted. Blue indicates that the centralized approach is better, and red indicates that the decentralized approach is better. In Figure 6.13b, the average number of targets in T_k divided by the total number of steps is plotted for the centralized method. A value of 1 indicates that each satellite has on average one target available for imaging at each time step. Finally, Figures 6.13c and 6.13d plot the percent difference in the number of unique imaged and downlinked targets.

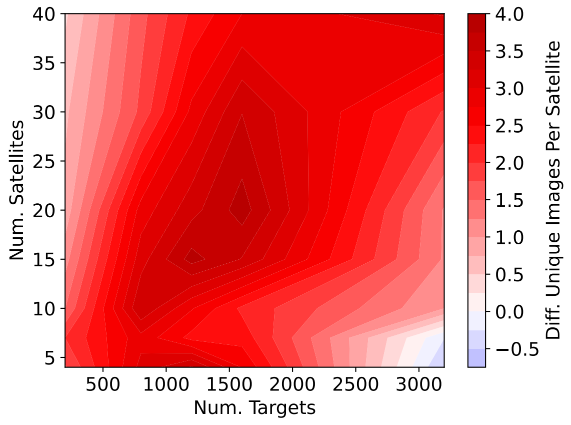
Several observations may be made regarding these plots. First, in Figure 6.13b, the centralized approach has almost one target on average per decision interval in the bottom right region,



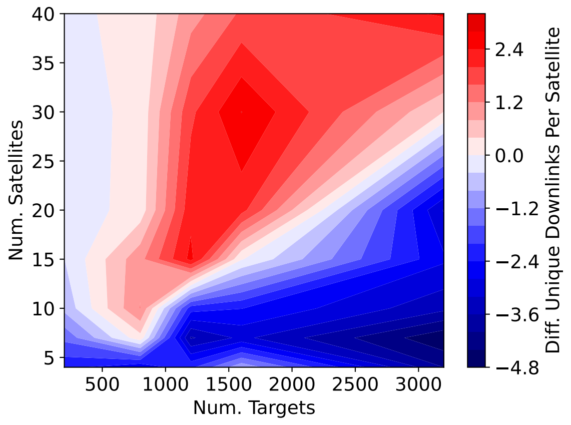
(a) Percent difference in reward.



(b) MIP target distribution statistics.



(c) Difference in unique images per satellite.



(d) Difference in unique downlinks per satellite.

Figure 6.13: Nominal and MIP target distribution comparison metrics. Blue indicates that the centralized approach is better, and red indicates that the decentralized approach is better.

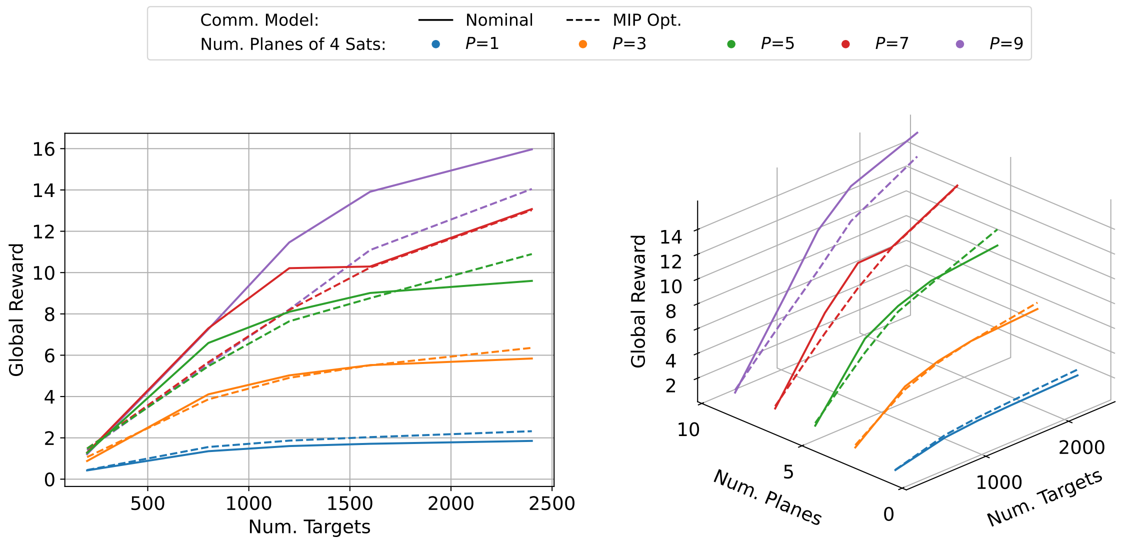
which occurs when small numbers of satellites and large numbers of targets are present. Furthermore, this region of high target assignment shows up in Figure 6.13a. When the centralized approach has almost one target available each decision-making interval, it performs better than the decentralized approach. An interesting observation can be made when low numbers of targets are available (< 1000). In this region, the centralized MIP approach results in < 0.5 targets distributed to each satellite at each step on average. However, the centralized MIP approach performs better than the decentralized approach, which seems counter to the previous argument. An explanation for this can be found in Figure 6.8. For low numbers of targets, the decentralized approach, even with free communication, results in a relatively high amount of duplication because there is more competition for a given ground target. The centralized approach ensures duplication is not an issue, so for very low numbers of targets the centralized approach outperforms the decentralized approach.

In Figure 6.13c, it is shown that the decentralized approach always results in more unique targets imaged. Two exceptions to this is in the aforementioned bottom right region, where the centralized distribution method has about the same number of uniquely imaged targets, and the left-hand region, where the centralized region has few targets distributed, but performs better. Finally, Figure 6.13d provides the rest of the story. In the areas where the percent difference in reward favors the centralized approach, the centralized approach is shown to have downlinked more targets on average. Intuitively, this makes sense because the largest component of reward is downlink.

6.4.2.2 Multi-Plane Results

The second set of ground target distribution experiments repeats the comparison between the centralized and decentralized distribution methods for Walker-delta constellations with multiple planes. The Walker-delta and ground target parameters are the same as those used in the communication experiment. The global reward comparing the two methods is provided in Figures 6.14. The same trends observed for the single plane experiments are shown here as well. For large constellations (≥ 9 planes, 36 satellites), the decentralized target distribution approach performs best for all numbers of global targets. For small constellations (≤ 1 plane, 4 satellites), the centralized distribution method performs better for almost all numbers of imaging targets. For constellations with 3-7 planes, the centralized approach performs best for large numbers of targets, but worse for small numbers of targets.

Metrics regarding the percent difference in reward, the average number of distributed targets per step, the difference in number of unique images, and difference in number of unique downlinks



(a) 2D view.

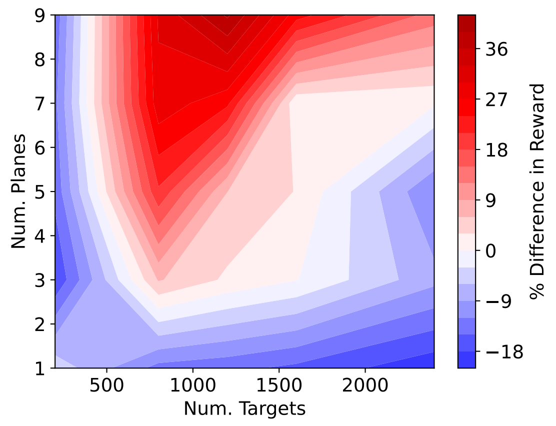
(b) 3D view.

Figure 6.14: A comparison of global reward for nominal vs. MIP target distribution methods.

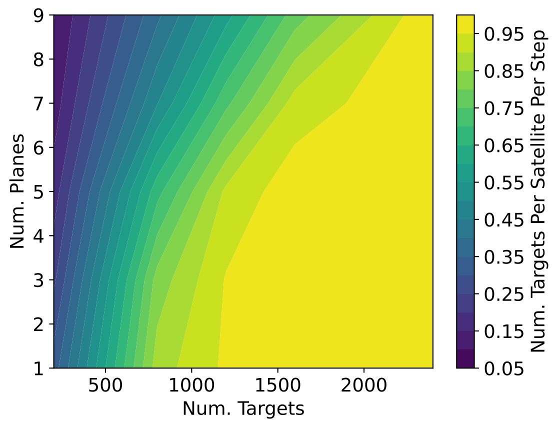
are plotted in Figure 6.15. The trends present in the single plane experiment are also present here. When the centralized distribution approach results in roughly one target distributed to each satellite at each decision-making interval, the centralized distribution method either performs the same or better than the decentralized distribution method. The centralized approach also outperforms the decentralized approach for low numbers of targets because it eliminates the issue of target duplication. In all other regions, the decentralized approach outperforms the centralized approach because of its ability to capture missed targets. Finally, the decentralized distribution method results in more unique targets imaged, but the centralized distribution method results in more unique targets downlinked in regions where there centralized target assignment averages are close to one or where duplication following the decentralized approach is relatively high.

6.4.2.3 Target Distribution Discussion

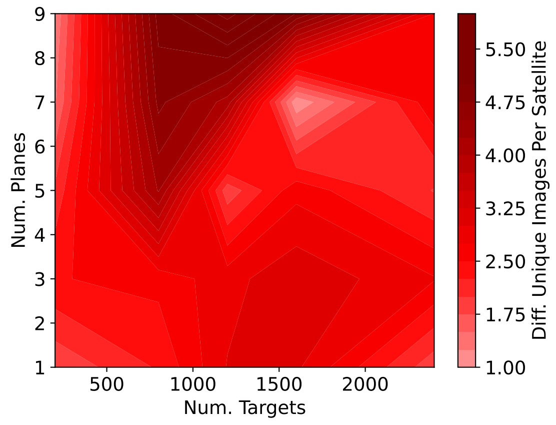
The difference in the performance between the two target distributions highlights the power of the approach taken in this work. The centralized method of target distribution typically only improves performance for constellations with small numbers of targets or constellations with small



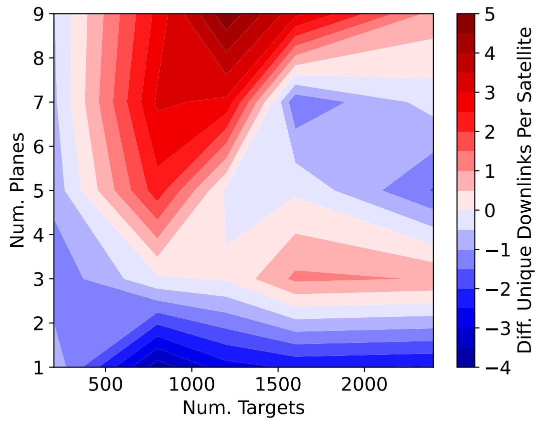
(a) Percent difference in reward.



(b) MIP target distribution statistics.



(c) Difference in unique images per satellite.



(d) Difference in unique downlinks per satellite.

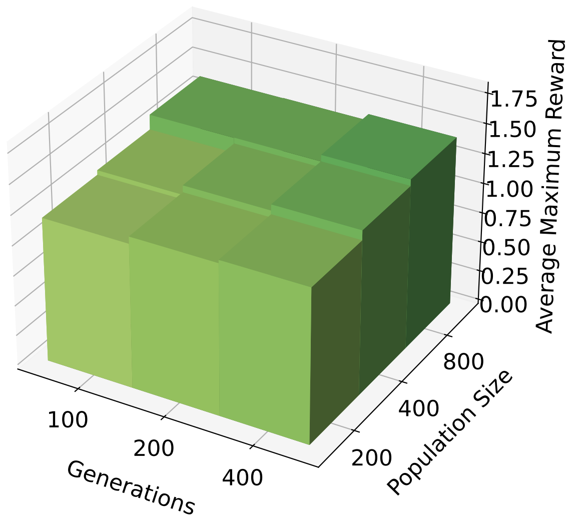
Figure 6.15: Nominal and MIP target distribution comparison metrics. Blue indicates that the centralized approach is better, and red indicates that the decentralized approach is better.

numbers of satellites. In larger constellations with many targets, the ordered access distribution algorithm performs better because it provides more flexibility in regard to which satellite collects the target. If one satellite must perform resource management activities and miss a collection opportunity, another satellite can collect the target later. This method is also more robust to additions to target lists from other satellites within the constellation. If the satellites in the constellation fly science detection algorithms that can detect new targets, the satellites can communicate the existence of the new targets to the other satellites in the constellation. In this case, the other satellites simply need to determine when they can access the new targets and add them to their target lists accordingly. This is a much simpler process than the centralized approach, which would need to re-optimize the target distribution problem.

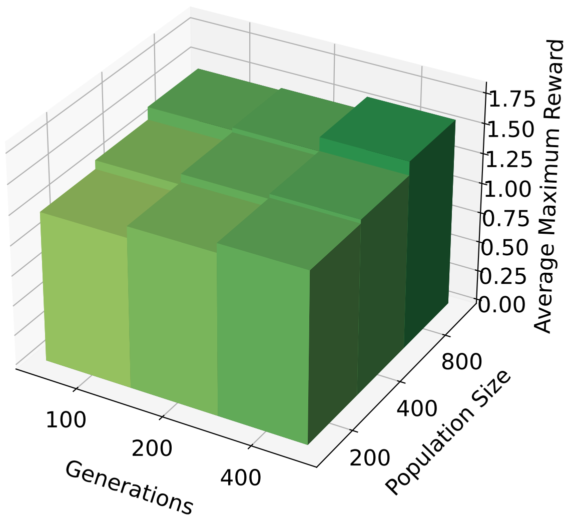
6.4.3 Genetic Algorithm Comparison

To compare the selected solution method to other state-of-the-art methods, a genetic algorithm is used to solve the MSAEO scheduling problem. For this experiment, two Walker-delta configurations are investigated. To keep the required computation to a minimum, each Walker-delta configuration contains only four spacecraft. With four spacecraft and $|U| = 3$, the size of the action space is $(3 + |U|)^4 = 1296$. In the first experiment, each of the four satellites are located in a single plane and have 3200 imaging targets available. In the second experiment, the four satellites are distributed among four planes and have 2400 targets available. The inclination of each satellite is 45 deg. Again, a phasing factor of 0 is utilized.

The results of the experiment are presented in Figure 6.16. The genetic algorithm is benchmarked for different numbers of population size and generations. Each bar in the figure is the average of the GA performance for 20 different trials, where each trial is a different initial condition. The maximum of the z-axis is set to the maximum average reward of the RL algorithm for four satellites in a single plane. This is done to provide a direct comparison between the two methods. The results here show that the GA, using the selected hyperparameters, performs worse than the RL methodology even though the GA is searching for a globally optimal solution. The GA likely needs more generations and a larger population size to converge to the globally optimal solution. However, this would come at a huge cost in terms of computation. For reference, the GA takes an average of 1.23 hours to complete 100 generations with a population size of 200 using 64 cores of an AMD Milan CPU with 240 GB of RAM. At 400 generations and a population size of 800, this increases to an average of 16.97 hours. The GA would need days of computation to converge to the solution found by the RL method for a single initial condition. For reference, the RL method takes 1-2 days to generate a neural network that can generalize to any initial condition.



(a) 4 satellites. Single plane. 3200 global targets.



(b) 4 satellites. 4 planes. 2400 global targets.

Figure 6.16: Average reward of the genetic algorithm under various hyperparameters. 20 trials each.

6.5 Conclusion

This chapter explores the application of single agent reinforcement learning to multi-satellite constellation operations. To explore the performance and scalability of this method, performance benchmarks are collected for different communication assumptions and different methods of target distribution in different Walker-delta constellation designs. Four communication models are tested: no communication, single degree line-of-sight communication, multi-degree line-of-sight communication, and free communication. The free communication model always outperforms the no communication model because satellites following the no communication model are constantly imaging and downlinking the same targets, receiving no reward for the duplication of efforts. The performance of the single- and multi-degree line-of-sight communication assumptions converge to either free communication or no communication, depending on how frequently the satellites may communicate. To test the impact on performance when coordination is utilized, two separate target distribution methods are also tested. The default target distribution method takes a decentralized approach where satellites are assigned targets based on their access to the imaging targets. The second target distribution method takes a centralized approach where a mixed integer program is formulated, and the targets are optimally distributed among the satellites based on their access. The centralized target distribution method performs better than the decentralized approach when A.) the centralized approach results in 0.9 - 1.0 targets distributed to each satellite at each decision-making interval on average and B.) there are few targets, which results in high amounts of duplication for the decentralized approach. In the majority of other cases, the decentralized approach is best because it allows satellites to share imaging targets. If one satellite misses a ground target because it must perform resource management activities, another satellite may image that target. Finally, the RL-based solution method is compared to the performance of a genetic algorithm. The RL-based solution method is shown to outperform the GA for a fraction of the computation cost.

Chapter 7

Small Body Science Scheduling Problem

7.1 Introduction

Missions to small bodies such as asteroids and comets present several challenges for planning and scheduling. First and foremost, epistemic uncertainty regarding the environment about small bodies necessitates the development of tools that can quickly adjust to the discovered environmental parameters upon arrival to the body. Second, large navigation uncertainties can lead to challenges in resource modeling and science operations. Either the uncertainty in task execution times and resource consumption must be handled explicitly in the scheduling algorithm or a buffer must be added to the end of every task to account for variations in execution time and resource consumption. Finally, the round-trip light-time delay can present challenges, especially during critical maneuvers such as Touch-and-Go (TAG). While these challenges are often addressed by work in autonomous guidance, navigation, and controls (GNC), planning and scheduling must be able to support rapid changes in the trajectory due to autonomous GNC.

On-board planning and scheduling has been implemented on several missions in recent decades. The ASPEN and CASPER systems developed by the Jet Propulsion Laboratory have been used in various forms for the Earth-Observing 1 mission [11, 88], IPEX mission [13], and even the Perseverance Rover [89, 90]. Up until this point, the focus of this dissertation is the application of reinforcement learning to Earth-observing satellite scheduling for on-board planning and scheduling. The EOS scheduling problems utilize a mode-based planning approach where devices on the spacecraft are turned on or off and the attitude guidance is set. The small body science

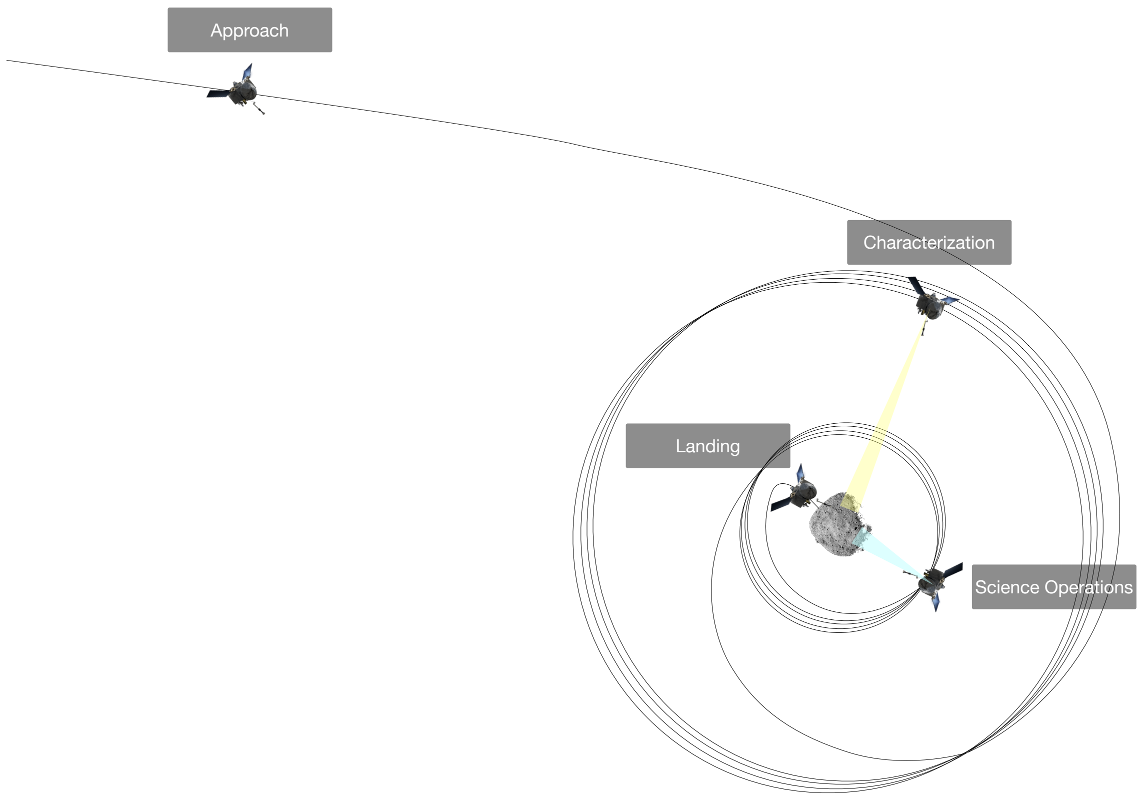


Figure 7.1: Small body mission phases.

operations problem follows a similar approach, but now adds translational guidance to the problem. The rest of this dissertation focuses on planning and scheduling for small body missions, which brings a new set of challenges. In the small body domain, deep RL has been applied to global mapping for shape modeling and target imaging. Chan and Agha-Mohammadi formulate a small body mapping problem as a partially-observable Markov decision process (POMDP) where the objective is to improve the quality of a map assembled using stereophotoclinometry (SPC) [91]. The authors apply the REINFORCE algorithm to generate policies over the belief-space, showing that the trained policies perform better than heuristic policies. Piccinin et al. formulate a global mapping problem for SPC as an MDP [92]. In this problem, the spacecraft enters an orbit about the body, and the decision-making agent determines whether or not to take an image. The authors compare Deep Q-Learning (DQN) and Neural Fitted Q (NFQ) learning, showing that these two algorithms outperform random and heuristic policies. Takahashi and Scheeres formulate a surface imaging problem about a small body as an MDP where the output of the policy is a change in elevation and a transfer time, which is fed into a two-point boundary value solver that generates a fuel-optimal control solution [93]. An extended Kalman filter is implemented to provide a state estimate to the two-point boundary value problem solver and decision-making agent. The authors apply Proximal Policy Optimization to train decision-making agents, showing how autonomous GNC technologies may be combined with reinforcement learning for surface imaging.

Past work has demonstrated how various proximity operations problems about small bodies may be formulated as (PO)MDPs and solved with reinforcement learning algorithms. However, these problem formulations typically fail to account for resource constraints such as on-board storage and power. Because on-board storage is not modeled, communication with the ground is typically left out of the problem formulations as well. Attitude guidance and control and its relation to the aforementioned resource constraints, particularly power, is also not considered. The addition of these aspects of the problem are important because they have serious implications for the learned policies. Furthermore, while many of these problem formulations add partial observability, the impact of partial observability on performance, particularly the quality of science observations, is not explored. It should also not be assumed that the navigation architecture supports continuous measurement updates. Instead, one should assume that the measurement update either requires communication with the ground or dedicated imaging for optical navigation, which means that the estimation error covariance should grow between navigation updates. To address these gaps in the literature, this work formulates a small body science proximity operations problem with on-board storage, power consumption and generation, data downlink, attitude guidance and control, and translational guidance and control. Various navigation assumptions are also explored, including directly observing the state (i.e. no partial observability), observing noisy measurements of the state directly, filtering with continuous measurement updates, and filtering with mode-based measurement updates. Part of this work is published in Reference [94].

Table 7.1: Small body mission phases.

	Approach	Characterization	Science Operations	Landing
Data Products	Body Ephemeris, Spin State, Preliminary Shape Model	Preliminary Science, Gravity Estimate, Improved Shape Model	Science Maps, Landing Site Images, Detailed Shape Model	Surface Science
Optical Navigation	Centroid-Based	Centroid-Based	Feature-Tracking	Feature-Tracking
Dynamics	Approach Trajectory	Hyperbolic Fly-bys	Orbital Motion, Inertial Waypoints, Low-Altitude Fly-bys	Descent & Ascent Trajectory
Rosetta Phases	Far Approach Trajectory	Close Approach Trajectory and Characterization	Global Mapping, Close Observation	Philae
OSIRIS-REx Phases	Approach	Preliminary Survey	Detailed Survey, Orbital B, Reconnaissance	Touch-And-Go

7.2 Small Body Proximity Operations Phases

Small body proximity operations may be described with several phases, each with its own objectives and data products. Each of these phases may be thought of as separate operations problems where the science and data products from one phase are utilized in the next. Past work in spacecraft autonomy for small body exploration has defined these mission phases in various ways [95, 96]. This chapter will provide its own summary for clarity. Because these phases are defined using concepts of operations from several different missions, the boundaries between them are fluid. Ashman et al. provide a detailed summary of the Rosetta operations phases [97], and Lauretta et al. provide a summary of the OSIRIS-REx operations phases [98]. The phases this work defines are A.) Approach, B.) Characterization, C.) Science Operations, D.) Landing. The characteristics of each phase are summarized in Table 7.1, and an example depiction of these mission phases are presented in Figure 7.1.

The first phase is the approach phase. During the approach phase, the spacecraft performs trajectory correction maneuvers to rendezvous with the asteroid. During this phase, a low fidelity shape model is constructed, a refined estimate of the spin state is gathered, and the ephemeris of the body is improved [96]. This phase is analogous to Rosetta's Far Approach Trajectory (FAT) Phase and OSIRIS-REx's Approach Phase. The second phase is typically a characterization phase. During this phase, the spacecraft enters the body's sphere of influence, performing hyperbolic flybys about the body. The shape model is improved, preliminary science data is gathered, and an estimate of the body's gravitational parameter is generated. This phase is analogous to Rosetta's Close Approach Trajectory (CAT) and Characterization Phase and OSIRIS-REx's Preliminary Survey Phase. Finally, the spacecraft enters the science operations phase, which may be decomposed further into more specific operations phases depending on the mission. This is when the detailed science campaign about the body begins, which is highly dependent on the mission. During this phase, the spacecraft either enters into a stable orbit about the body, transfers between or holds a position at an inertial waypoint(s), or performs low-altitude fly-bys about the body. This also marks the transition from centroid-based optical navigation to feature-tracking optical navigation due to the spacecraft's proximity to the body. This phase typically includes some sort of mapping to build temperature maps, reflectance maps, and identify candidate landing sites. In the case of Rosetta, the Global Mapping and Close Observation Phases fall into this category. In the case of OSIRIS-REx, the Detailed Survey, Orbital B, and Reconnaissance Phases fall into this category. The final phase of proximity operations is often some sort of landing phase. In the case of Rosetta this includes the landing of the Philae lander, and in the case of OSIRIS-REx this includes the Touch-and-Go phase.

7.3 Problem Statement

This chapter formulates a small body science operations problem where a spacecraft maneuvers between waypoints defined in the sun-asteroid Hill frame, performing science activities while managing on-board resources such as power and data storage. The objective is to maximize the number of targets imaged and downlinked and the amount of mapping performed and downlinked. Therefore, there are two simultaneous science objectives - spectroscopy mapping and high-resolution target imaging. For the spectroscopy mapping, there are $j = 3$ separate maps that must be collected, one at each of the following local solar times: {6 PM, 2 PM, 10 AM}. Each map is represented by a set of $k = 500$ points, M_j, evenly distributed on the surface of the body,

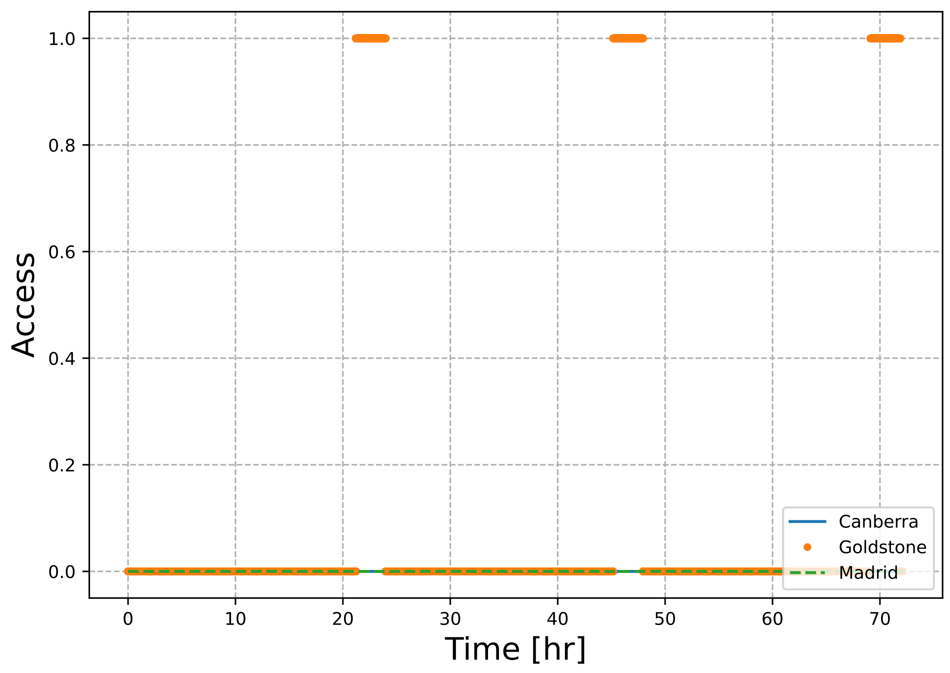


Figure 7.2: Deep space network access with temporal constraints over three days of operations.

where j is the map number. These points are generated using a Fibonacci lattice to ensure equal coverage of the body. The high resolution imagery is represented by a set of surface targets that are referred to as T. The spacecraft has pre-planned access with the deep space network (DSN) once every 24 hours. An example of the DSN access over three days of operations is provided in Figure 7.2.

In addition to the science objectives, the spacecraft must avoid collision with the body as it maneuvers between waypoints. The waypoints are defined in the sun-asteroid Hill frame at three radii away from the center of the body (approximately 750 m) using spherical coordinates. Each waypoint w is represented with the nominal radius $r_w = 750$ m, a polar angle ψ, and an azimuth angle θ. A diagram of the sun-asteroid Hill frame and definition of the waypoints is provided in Figure 7.3.

This small body science operations problem is most closely related to the OSIRIS-REx Detailed Survey Phase. However, this work adds additional waypoints such that maneuvers are not only performed in northern and southern regions, moves the spacecraft closer to the body, and

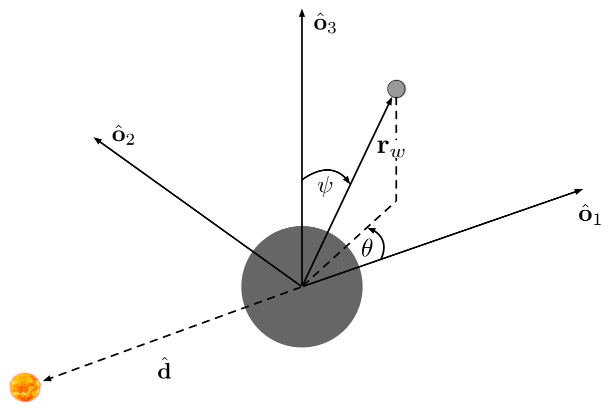
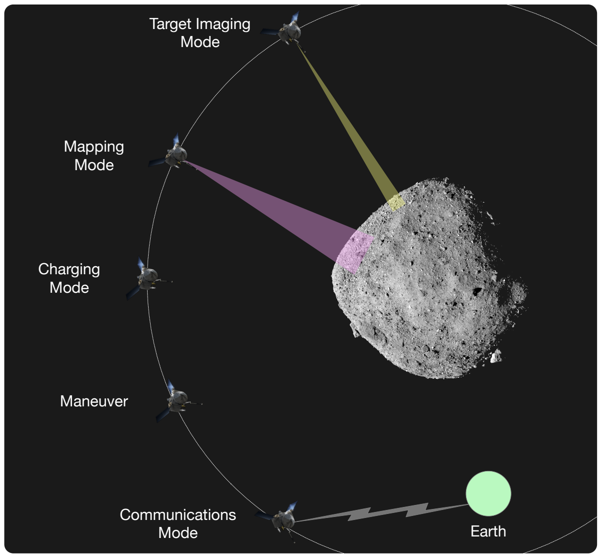


Figure 7.3: Sun-asteroid Hill frame.

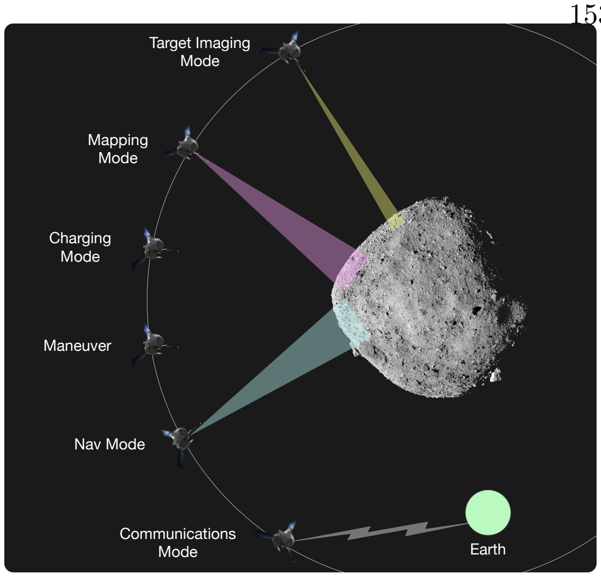
increases the half field-of-view of the mapping instrument. This is primarily done to reduce the amount of time the spacecraft is coasting in regions where there is no science value, decreasing the simulation time required to complete the mapping campaign. To complete the mission, the spacecraft enters different modes of operation. These spacecraft modes abstract the continuous, low-level behaviors of the spacecraft (i.e. attitude guidance and control, instrument status, etc.) into discrete modes of operations. These modes are shown in Figure 7.4. In the figure on the left, the states observed by the decision-making agent are either the true states of the spacecraft observed directly, noisy measurements of the spacecraft state, or a belief state from an extended Kalman filter (EKF) receiving continuous measurements. In the figure on the right, a navigation mode is added to the action space. During the navigation mode, the spacecraft receives measurements and the measurement update state of the EKF is utilized to reduce the state-error covariance and improve the navigation solution. In all other modes, the navigation solution decays in quality and the state-error covariance grows because only the prediction step of the EKF is utilized.

7.3.1 Dynamics

The position and velocity of the spacecraft, ${}^{\mathcal{O}}\mathbf{r}$ and ${}^{\mathcal{O}}\dot{\mathbf{r}}$, are expressed in the sun-asteroid Hill frame, $\{\mathcal{O} : \hat{\mathbf{o}}_1, \hat{\mathbf{o}}_2, \hat{\mathbf{o}}_3\}$, which is a convenient coordinate frame for this problem due to the illumination requirements for mapping. The derivation of the relative dynamics may be found



(a) Nominal.



(b) Navigation mode included.

Figure 7.4: Small body science operations problem operational modes.

in work from Scheeres [99] and Takahashi [100]. A brief description is also provided here. The dynamics described in this section are utilized within the extended Kalman filter and continuous feedback control law used for translational control about the body. Basilisk models the dynamics of the spacecraft as fully coupled multibody dynamics, making no simplifying assumptions regarding the relative motion of the spacecraft, the asteroid, and the sun. However, both Basilisk and the dynamics described in this section utilize a cannonball solar radiation pressure (SRP) model.

The equations of motion for the spacecraft in proximity to the small body are given as follows:

$$\ddot{\mathbf{r}} = -\ddot{F}\tilde{\hat{o}}_3\mathbf{r} - 2\dot{F}\tilde{\hat{o}}_3\dot{\mathbf{r}} - \dot{F}^2\tilde{\hat{o}}_3\tilde{\hat{o}}_3\mathbf{r} + \frac{\partial U_g}{\partial \mathbf{r}} + \frac{\partial U_s}{\partial \mathbf{r}} + \mathbf{a}_{\text{srp}}. \tag{7.1}$$

$\dot{F}$ is the first time derivative of the true anomaly:

$$\dot{F} = \sqrt{\mu_{\text{sun}}/[A(1-E)^2]^3}(1 + E\cos F)^2, \tag{7.2}$$

$\ddot{F}$ is the second time derivative of the true anomaly:

$$\ddot{F} = -2E\sqrt{\mu_{\text{sun}}/[A(1-E)^2]^3}\sin F(1 + E\cos F)\dot{F}, \tag{7.3}$$

The gravitational parameter of the Sun is given by μ_{sun}, the semi-major axis of the asteroid is given by A, and the eccentricity of the asteroid's orbit is E.

A point-mass gravity model is utilized for the asteroid, with the derivative of the gravitational potential given as

$$\frac{\partial U_g}{\partial \mathbf{r}} = -\frac{\mu_{\text{ast}}}{r^3}\mathbf{r}, \tag{7.4}$$

where μ_{ast} is the gravitational parameter of the asteroid.

The gravity of the sun is modeled as a third-body perturbation, with the derivative of the gravitational potential given as

$$\frac{\partial U_s}{\partial \mathbf{r}} = \frac{\mu_{\text{sun}}(3\hat{\mathbf{d}}\hat{\mathbf{d}}^T - [I_{3\times3}])}{d^3}\mathbf{r}, \tag{7.5}$$

where $\hat{\mathbf{d}}$ is the direction of the sun.

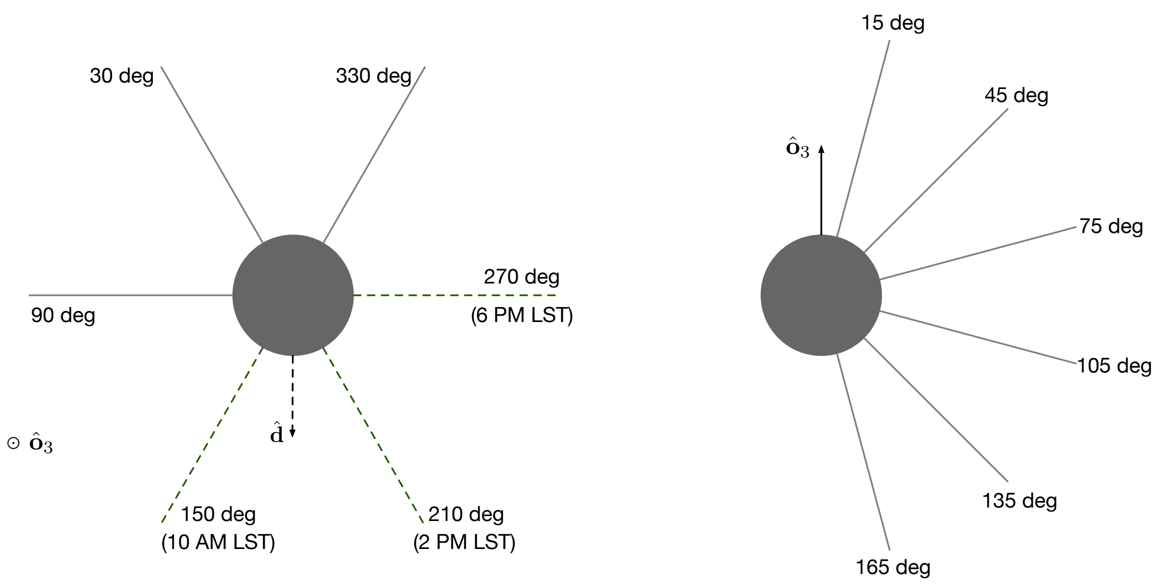
Finally, a cannonball SRP model is utilized. The acceleration due to SRP is given as

$$\mathbf{a}_{\text{srp}} = \frac{P_0(1+\rho)A_{\text{sc}}(1AU)^2}{M_{\text{sc}}d^2}\hat{\mathbf{d}}, \tag{7.6}$$

where ρ is the surface reflectivity, A_{sc} is the surface area of the spacecraft, M_{sc} is the mass of the spacecraft, and P_0 is the solar flux at 1 AU. The values for ρ (0.4) and P_0 ($4.56 \times 10^{-6}\text{Nm}^{-2}$) are taken from Takahashi and Scheeres [100].

7.3.2 Waypoint Maneuvering

The waypoints the spacecraft maneuvers between are defined in the sun-asteroid Hill frame, $\mathcal{O}$. The spacecraft maneuvers between or holds its position at specific waypoints, performing the tasks in Figure 7.4 as the asteroid rotates beneath it. The waypoints are evenly distributed across six polar and azimuth angles, as shown in Figure 7.5, numbering 36 in total. In Figure 7.5a, the dotted lines represent the local solar times where spectroscopy mapping may take place. The $\hat{\mathbf{d}}$ vector denotes the direction of the sun. There are three maps in total that must be collected. In Figure 7.5b, the various polar angles are displayed. Mapping may occur at any of these polar angles if the spacecraft is at an azimuth angle associated with the local solar time for a map. During the



(a) Azimuth angles for waypoints (θ). (b) Polar angles for waypoints (ψ).

Figure 7.5: Spherical coordinates of waypoints. Dotted lines represent the local solar time of the three maps.

Detailed Survey Phase, OSIRIS-REx had seven total maps to collect, each at a specific local solar time. Furthermore, the mapping had to take place at a relatively narrow band of polar angles. This work selects three maps at specific local solar times and removes the narrow polar angle requirement to maintain minimal simulation time.

A continuous feedback control law is implemented to maneuver the spacecraft between the various waypoints using the methodology described in Chapter 14 of Schaub and Junkins [73]. The control acceleration is computed with the following equation,

$$\mathbf{u} = -(\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_{ref})) - [K_1]\Delta\mathbf{x}_1 - [K_2]\Delta\mathbf{x}_2 \tag{7.7}$$

where $\mathbf{f}(\mathbf{x})$ is given in Equation 7.1. The commanded thrust is computed by dividing this acceleration by the mass of the spacecraft. In the case where noisy measurements of the spacecraft position and velocity are utilized, a deadband is placed around the state vectors such that the control law does to expend all of it's ΔV budget correcting for small errors due to noise. The deadband is not required in the event that the state is known perfectly or the EKF is utilized to estimate the state.

This feedback control law provides no guarantees on fuel optimality. Future work should consider the use of a Lambert solver to compute a fuel-optimal two-burn solution. However, the feedback control law fulfills the function of a control solution from one waypoint to another while simultaneously performing station keeping activities. Furthermore, the total ΔV can be computed and compared to the ΔV budget. For the purposes of planning and scheduling, this solution is sufficient.

7.3.3 State Estimation

Next, the mission simulation fidelity is enhanced with the inclusion of an EKF to estimate the state of the spacecraft $\mathbf{x} = [{}^{\mathcal{O}}\mathbf{r};\ {}^{\mathcal{O}}\dot{\mathbf{r}}]$. An extended Kalman filter produces a state estimate for a dynamical system by predicting the state of the system through integrating equations of motion and updating this prediction with measurements from the environment [101]. Navigation solutions for small body missions typically use some combination of radiometric ground tracking measurements from Earth-based sensors like the DSN and optical measurements in proximity about the small body that are matched with landmark maps of the surface of the body, which are used to provide the relative navigation solution. Batch filtering is the state-of-the-art for small body missions [102, 103, 104, 105]. An iterative process between the orbit determination (OD) and optical navigation (OPNAV) teams occurs where the OD team updates the spacecraft trajectory and the OPNAV team updates the landmark maps. This is an intensive ground-based process, and the navigation solution for the next epoch is sent back up to the spacecraft for the next epoch of operations. Sequential state estimation solutions like extended or unscented Kalman filters are popular in the literature for autonomous spacecraft operations as they can continually produce a state estimate on-board the spacecraft [41, 106, 107]. A benefit of such filters is that they do not require multiple iterations to converge to a solution, which a batch filter does.

The algorithm for the EKF is provided in Algorithm 11. The initial state of the EKF is initialized with a small amount of error added to the truth state sampled from uniform distributions

Algorithm 11 Extended Kalman filter for small body navigation.

```
1: Initialize $i = 1$, $t_{i-1} = t_0$, $\hat{\mathbf{x}}^+_{i-1} = \hat{\mathbf{x}}_0$, $P^-_{i-1} = P_0$
2: for iteration $1 : N$
3:     Propagate dynamics:
4:         $\dot{\mathbf{x}}(t) = \mathbf{f}(\hat{\mathbf{x}}^+_{i-1}, \mathbf{u}_{i-1}, t)$
5:         $A(t) = \left.\frac{d\mathbf{f}}{d\mathbf{x}}\right|\hat{\mathbf{x}}$
6:         $\dot{\Phi}(t, t_{i-1}) = A(t)\Phi(t, t_{i-1})$
7:         Compute $\hat{\mathbf{x}}^i_i$ and $\Phi(t_i, t_{i-1})$ using RK4 integration
8:     Update covariance:
9:         $P^-_i = \Phi(t_i, t_{i-1})P^+_{i-1}\Phi^T(t_i, t_{i-1}) + Q(t_i, t_{-1})$
10:    if new measurements
11:        Read measurements, $\mathbf{y}_i$
12:        Compute measurement residuals and Kalman gain
13:            $\mathbf{r}_i = \mathbf{y}_i - \mathbf{h}(\hat{\mathbf{x}}^-_i, t_i)$
14:            $K_i = P^-_i H^T_i (H_i P^-_i H^T_i + R)^{-1}$, where $H_i = \left.\frac{d\mathbf{h}}{d\mathbf{x}}\right|\hat{\mathbf{x}}^-_i$
15:        Perform measurement update
16:            $\hat{\mathbf{x}}^+_i = \hat{\mathbf{x}}^-_i + K_i\mathbf{r}_i$
17:            $P^+_i = (I - K_iH_i)P^-_i(I - K_iH_i)^T + K_iRK^T_i$
18:    else
19:        $\hat{\mathbf{x}}^+_i = \hat{\mathbf{x}}^-_i$
20:        $P^+_i = P^-_i$
21:        $i = i + 1$
```

of $\mathcal{U}[-5, 5]$ m and $\mathcal{U}[-0.01, 0.01]$ m/s for the position and velocity of the spacecraft in the sun-asteroid Hill frame. The first step of the algorithm is to propagate the dynamics of the system forward in time. The dynamics are propagated using a fourth-order Runge-Kutta (RK4) integrator. The state transition matrix, $\Phi(t, t_{i-1})$, is computed by integrating the linearized dynamics of the system, which are computed using the Jacobian of the dynamics, $A(t) = \left.\frac{df}{dx}\right|\hat{\mathbf{x}}$. The state transition matrix is then used to propagate the covariance forward in time. The covariance is propagated using the following equation,

$$P^-_i = \Phi(t_i, t_{i-1})P^+_{i-1}\Phi^T(t_i, t_{i-1}) + Q(t_i, t_{i-1}), \tag{7.8}$$

where $Q(t_i, t_{i-1})$ is the process noise covariance. The process noise covariance is computed using the state noise compensation (SNC) algorithm [108]. The process noise covariance at any time t_i

Table 7.2: SimpleNav parameters.

Parameter	Position	Velocity
σ	5 m	0.001 m/s
Walk Bounds	1 m	0.001 m/s

is given as:

$$Q(t_i, t_{i-1}) = \sigma_u^2 \begin{bmatrix} \dfrac{\Delta t^3}{3}[I_{3\times 3}] & \dfrac{\Delta t^2}{2}[I_{3\times 3}] \\ \dfrac{\Delta t^2}{2}[I_{3\times 3}] & \Delta t[I_{3\times 3}] \end{bmatrix} \tag{7.9}$$

The diffusion coefficient σ_u^2 was experimentally tuned and is set to 10^{-11} m/s^2.

After the propagation step, the algorithm checks to see if there are any new measurements. If there are new measurements, the measurement update step is performed. The measurement residuals are computed as:

$$\mathbf{r}_i = \mathbf{y}_i - \mathbf{h}(\hat{\mathbf{x}}_i^-, t_i), \tag{7.10}$$

where $\mathbf{y}_i$ is the measurement vector and $\mathbf{h}(\hat{\mathbf{x}}_i^-, t_i)$ is the measurement model. Basilisk's `simpleNav` module is used to provide measurements to the extended Kalman filter at 0.5 Hz. This module utilizes a second-order Gauss-Markov error model to provide realistic measurement error. While `simpleNav` does not capture several of the intricacies of small body navigation measurements (i.e. scale invariance or a dependency on lighting conditions) it provides a reasonable approximation of measurement error beyond additive white Gaussian noise. The standard deviations for the white noise component of the measurement error model and the walk bounds of the Gauss-Markov process are provided in Table 7.2. Because the measurement model is simply the states of the spacecraft with noise and random walk added, the Jacobian of the measurement model is simply the identity matrix. After the measurement residuals are computed, the Kalman gain is computed as:

$$K_i = P_i^- H_i^T (H_i P_i^- H_i^T + R)^{-1}, \tag{7.11}$$

The measurement update step is then completed with the following two equations, where the updated state is computed as:

$$\hat{\mathbf{x}}_i^+ = \hat{\mathbf{x}}_i^- + K_i \mathbf{r}_i, \tag{7.12}$$

and the covariance is updated as:

$$P_i^+ = (I - K_i H_i) P_i^- (I - K_i H_i)^T + K_i R K_i^T, \tag{7.13}$$

where R is the measurement noise covariance matrix.

The entire process repeats until the end of the simulation. If no measurements are provided after the propagation step, the algorithm simply propagates the state and covariance forward in time and writes these out as messages, which are read by the decision-making agent. An example of the position and velocity error and covariance bounds with periodic navigation updates are provided in Figure 7.9.

7.4 Markov Decision Process Formulation

The small body science operations problem is formulated as a Markov decision process. This section describes each of the components of the MDP formulation.

7.4.1 State Space

The state space of the small body science scheduling problem is given as:

$$\mathcal{S} = \mathcal{S}_{\text{sc}} \times \mathcal{S}_{\text{asteroid}} \times \mathcal{S}_{\text{maps}} \times \mathcal{S}_{\text{targets}} \times \mathcal{S}_{\text{DSN}} \tag{7.14}$$

The state returned to the decision-making agent at decision interval i is defined as $s_i \in S$:

$$\begin{aligned} s_i = (&{}^{\mathcal{O}}\mathbf{r}_{\text{sc}},\ {}^{\mathcal{O}}\mathbf{v}_{\text{sc}},\ {}^{\mathcal{O}}\mathbf{r}_{t_{\text{nearest}}},\ {}^{\mathcal{O}}\mathbf{r}_{w_{\text{ref}}},\ {}^{\mathcal{O}}\mathbf{r}_{w_{\text{prev}}}, \cdots \\ &\text{num. imaged targets, num. downlinked targets, map regions, battery}, \cdots \\ &\text{eclipse, buffer, } \Delta V \text{ consumed, ground station indicator}). \end{aligned} \tag{7.15}$$

Geometric information is included in the state space to capture the spatial relationship between the science objectives and the spacecraft. It can also provide information on resource management states and the risk of collision. These states include the spacecraft position and velocity

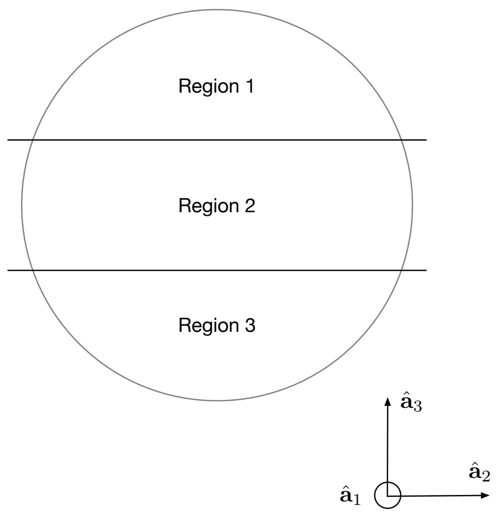


Figure 7.6: Map regions.

(${}^{\mathcal{O}}\mathbf{r}_{\text{sc}}$ and ${}^{\mathcal{O}}\mathbf{v}_{\text{sc}}$), position of the nearest imaging target (${}^{\mathcal{O}}\mathbf{r}_{t_{\text{nearest}}}$), position of the current waypoint (${}^{\mathcal{O}}\mathbf{r}_{w_{\text{ref}}}$), and position of the previous waypoint (${}^{\mathcal{O}}\mathbf{r}_{w_{\text{prev}}}$). These states are all expressed in the sun-asteroid Hill frame, $\mathcal{O}$, a convenient expression for this problem given that one of the primary science objectives is mapping at specific local solar times.

Several observations are also included to provide a measure of science objective completion. The number of imaged and downlinked targets in T are included in the state space. For each map M_j, the mapping points are partitioned into three equally sized groups based on the value of the z-component of the body-fixed position of the mapping points. The body frame of the asteroid is defined as $\mathcal{A} : \{\hat{\mathbf{a}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{a}}_3\}$. The three regions are displayed in Figure 7.6. This state provides the agent information on which regions still need to be mapped. Because this work assumes the rotation pole of the body is aligned with the orbit normal, this state representation is sufficient. However, varying the rotation pole of the body may require a more sophisticated state representation.

Finally, several states are included to retain information on resource constraints and safety. The data stored in the buffer and ground station indicator provide state information for the on-board data system. The battery charge and eclipse indicator provide information for the purposes of power management. The available ΔV state indicates how much fuel the spacecraft has available to use.

Each state is normalized to a range of approximately [-1, 1]. The spacecraft position, position of the nearest imaging target, and position of the current and previous waypoint are all normalized by the radius of the body. The velocity of the spacecraft in the sun-asteroid Hill frame, however, is not normalized because the velocity at the times observations are returned typically falls within a range of [-1, 1]. The number of imaged and downlinked targets, the mapped regions, and resource states are all normalized by their respective max values such that they are within a range of [0, 1].

In the event that the navigation mode is added to the action space and the agent acts using the belief state produced by the EKF, the estimate of the position and velocity of the spacecraft in the sun-asteroid Hill frame are used instead of direct observations of the state. Additionally, the log of the diagonal of the covariance matrix divided by five is added provided as an observation: $\log_{10}(\text{diag}(P))/5$. This provides information on the quality of the navigation solution that is normalized to a range of [-1, 1] for $10^{-5} \leq P \leq 10^{5}$.

7.4.2 Action Space

A mode-based planning approach is taken in the action space. A spacecraft mode turns certain models on or off and sets the attitude reference for a prescribed amount of time, abstracting continuous low-level behavior into higher-level abstractions of spacecraft behavior. While this does place limitations on the ability of the decision-making agent to specify precise timing for the duration of modes, it reduces the complexity of the planning problem, trading performance for tractability. An action space $\mathcal{A}$ is constructed for the small body science scheduling problem that allows the decision-making agent to collect and downlink science data, manage its resources, and transition between waypoints:

$$\mathcal{A} = \{\text{Charge, Waypoint 1}, \cdots, \text{Waypoint 8, Map, Image, Downlink, Nav Update}\}. \tag{7.16}$$

Each mode lasts for 2,000 seconds, with the exception of the mapping mode and optional navigation mode. The mapping mode lasts for 4,000 seconds, which is approximately one quarter of a full revolution of the body about its rotation pole. The navigation mode only lasts for 1,000 seconds. A detailed description of the action space is provided by the bulleted list below:

- **Charge:** The spacecraft points its solar panels at the sun, turning off all instruments and transmitters to recharge the batteries.

- **Waypoint Actions:** The spacecraft targets one of the eight neighboring waypoints, turning off all instruments and transmitters during the duration of the maneuver mode. The eight neighboring waypoints are defined as follows:

  * $\psi_{\text{ref}} = \psi_{\text{ref}} + 30^o$, $\theta_{\text{ref}} = \theta_{\text{ref}}$
  * $\psi_{\text{ref}} = \psi_{\text{ref}} + 30^o$, $\theta_{\text{ref}} = \theta_{\text{ref}} + 60^o$
  * $\psi_{\text{ref}} = \psi_{\text{ref}}$, $\theta_{\text{ref}} = \theta_{\text{ref}} + 60^o$
  * $\psi_{\text{ref}} = \psi_{\text{ref}} - 30^o$, $\theta_{\text{ref}} = \theta_{\text{ref}} + 60^o$
  * $\psi_{\text{ref}} = \psi_{\text{ref}} - 30^o$, $\theta_{\text{ref}} = \theta_{\text{ref}}$
  * $\psi_{\text{ref}} = \psi_{\text{ref}} - 30^o$, $\theta_{\text{ref}} = \theta_{\text{ref}} - 60^o$
  * $\psi_{\text{ref}} = \psi_{\text{ref}}$, $\theta_{\text{ref}} = \theta_{\text{ref}} - 60^o$
  * $\psi_{\text{ref}} = \psi_{\text{ref}} + 30^o$, $\theta_{\text{ref}} = \theta_{\text{ref}} - 60^o$

- **Map:** The spacecraft turns on its mapping instrument, collecting the map of whichever map region it is currently in.

- **Image:** The spacecraft turns on its imaging instrument, collecting an image of the nearest target.

- **Downlink:** The spacecraft turns on its transmitter, downlinking the data in the buffer to the DSN.

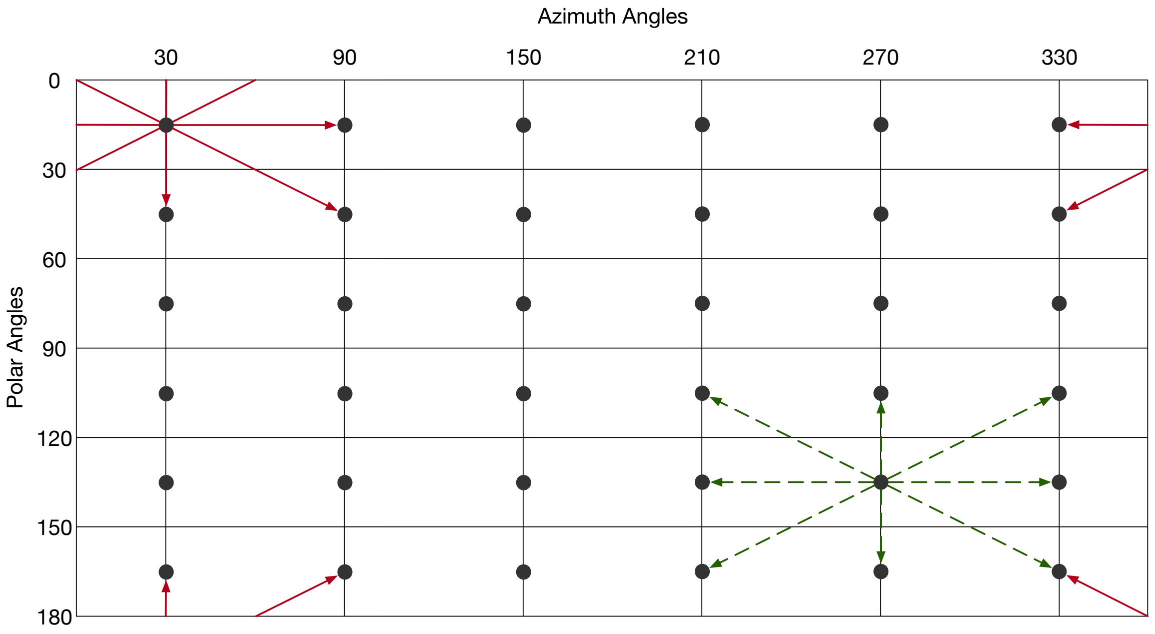


Figure 7.7: Waypoint reference transitions.

- **Navigation Update:** Only used in some experiments, the spacecraft begins collecting measurements to improve the state estimate.

In the charging mode, the spacecraft turns off all instruments and the transmitter and points the solar panels at the sun to charge the battery. The action space also includes eight separate waypoint reference change actions. When a waypoint reference change action is taken, the current waypoint reference $w_{\text{ref}} = \{\psi_{\text{ref}}, \theta_{\text{ref}}\}$ changes to the selected adjacent waypoint reference. If one of these modes is selected, the last time the waypoint was changed is checked to see if a new waypoint can be selected. The current waypoint does not change unless 8,000 seconds have passed since the last switch to ensure convergence to the current waypoint. After each change, the new waypoint polar and azimuth angles are checked to ensure they are wrapped to the appropriate ranges, $\theta \in [0, 360]$ deg and $\psi \in [0, 180]$ deg. An example of this is provided in Figure 7.7. The nominal transitions are shown in the dotted green line. Wrapped transitions are shown in the solid red line.

In the mapping mode, the spacecraft points the mapping instrument at the asteroid. Data is collected in the on-board storage unit, and only the portion of the map collected within requirements

Table 7.3: Science requirements

Imaging	
Elevation	60^o
Attitude Error Norm	0.1 rad
Mapping	
Elevation	45^o
Instrument Half-FOV	22.5^o
Azimuth Angle Tolerance	1^o

is considered mapped. Mapping requirements are provided in Table 7.3. Note that the requirement regarding the azimuth angle associated with the local solar time is 1 deg. At the nominal waypoint radius of 750 meters, this translates to roughly 6.5 meters of positional tolerance on either side of the azimuth angle. This is a very tight requirement. If the navigation mode is utilized in the action space, the spacecraft must periodically improve its state estimate with measurements to reduce the state error below this threshold.

In the imaging mode, the spacecraft points the imager at the nearest target and attempts to take an image of the target. The image is collected if the spacecraft is within the elevation and range requirements of the target image. In the downlink mode, the spacecraft points the transmitter in the direction of the Earth. Data is downlinked once the spacecraft is within elevation and range requirements of the DSN and the prescribed downlink time occurs.

Finally, the navigation mode is utilized in some experiments to provide a more realistic navigation update. In this mode, the spacecraft points an imager at the asteroid to simulate the use of feature-tracking optical navigation. The state estimate is improved, and the state-error covariance decreases.

7.4.3 Reward Function

The reward function $R(s_i, a_i, s_{i+1})$ is a piecewise function of the current state, action, and next state. The reward function builds off the reward functions designed for the agile EOS scheduling problem, but adds mapping and additional failure conditions. The constant F scales the failure penalty, the constant A scales the imaging bonus, the constant B scales the mapping bonus, the constant C scales the image target downlink component, and the constant D scales the mapping downlink component. The reward at state i is given by:

$$r_i = \begin{cases} -F & \text{if failure} \\ \frac{A}{|T|} H(c_j) & \text{if } \neg\text{failure} \wedge a_i \text{ is image} \\ \frac{B}{3|M|} \sum_j^3 \sum_k^{|\mathbf{M}_j|} H(m_{j,k}) & \text{if } \neg\text{failure} \wedge a_i \text{ is map} \\ \frac{C}{|T|} \sum_j^{|\mathbf{T}|} H(d_j) + \frac{D}{3|M|} \sum_j^3 \sum_k^{|\mathbf{M}_j|} H(f_{j,k}) & \text{if } \neg\text{failure} \wedge a_i \text{ is downlink} \\ 0 & \text{otherwise} \end{cases} \tag{7.17}$$

If the agent fails, a failure penalty of -F is returned and the episode terminates. The failure condition is true if the spacecraft expends all charge in the battery, overfills the data buffer, exceeds the ΔV budget, or collides with the body. Mathematically, this is represented as with Equation 7.18, where z is the normalized charge of the battery and b is the normalized data buffer level.

$$\text{failure} = (z == 0 \ \vee \ b \geq 1 \ \vee \ \text{any}(||^{\mathcal{H}}\mathbf{r}_{\text{s/c}}|| \leq \mathbf{r}_{\text{ast}}) \ \vee \Delta V \geq \Delta V_{\text{budget}}) \tag{7.18}$$

The same function $H(x_j)$ utilized in the SSAEO scheduling problem is again used here to check if the state variable x is false at step i and true at step $i+1$, returning 1 if these conditions are met.

$$H(x_j) = 1 \text{ if } \neg x_{j_i} \ \wedge \ x_{j_{i+1}} \tag{7.19}$$

The variable c_j represents whether or not target j has been imaged. If the imaging mode is initiated and a failure does not occur, target j is checked to determine if it was imaged for the first time. This reward component is normalized by the total number of targets and scaled by the constant A.

The variable $m_{j,k}$ represents whether or not mapping point k for map number j has been mapped. If the mapping mode is initiated and a failure does not occur, all map points are checked to determine if they were collected for the first time or not. The summation of this reward is normalized by $3|M|$ such the total possible reward for this component totals to the constant B.

The variable d_j represents whether or not target j has been downlinked, and the variable $f_{j,k}$ represents whether or not mapping point k for map number j has been downlinked. Both the set of targets and all map points are looped through to determine if they have been downlinked for the first time or not. Both the imaging and mapping components are multiplied by the constants C and D and divided by the total number of targets or mapping points.

7.4.4 Transition Function

Due to the continuous dynamics of the small body proximity operations science problem, it is difficult to construct a transition function with conditional probabilities that accurately captures state transitions. The transition function is instead represented by a generative model $G(s_i, a_i)$ given in Equation 7.20. The decision-making agent passes the action to the generative model, which turns on or off different flight software modes, sets the attitude reference, sets the reference for translational guidance, and determines behavior of the navigation system. The Basilisk simulation then integrates the simulation forwards in time, and the generative model returns a new state s_{i+1} and reward r_i by integrating equations of motion forwards in time.

$$s_{i+1},\ r_i = G(s_i, a_i) \tag{7.20}$$

The Basilisk astrodynamics software architecture [109] is used to construct the simulation, which models the complex behavior of the spacecraft and environment. The Basilisk simulation is wrapped within a Gymnasium environment. The Gymnasium environment provides a standard interface for the agent to interact with the Basilisk simulation. The details of this simulation are provided in the next section.

7.5 Simulation Architecture

7.5.1 Basilisk Simulation Overview

A Basilisk simulation is implemented to serve as the generative transition function for the MDP. In Figure 7.8, the task groupings and modules in the Basilisk simulation are provided.

Table 7.4: Basilisk model and task status in different modes.

	Modes				
Basilisk Tasks & Models	Charge	Waypoint Change	Map	Image	Downlink
Sun-Pointing Task	Enabled	Enabled	Disabled	Disabled	Disabled
Earth-Pointing Task	Disabled	Disabled	Disable	Disabled	Enabled
Location-Pointing Task	Disabled	Disabled	Disabled	Enabled	Disabled
Map-Pointing Task	Disabled	Disabled	Enabled	Disabled	Disabled
MRP Control Task	Enabled	Enabled	Enabled	Enabled	Enabled
Waypoint Control Task	Enabled	Enabled	Enabled	Enabled	Enabled
Mapping Task	Disabled	Disabled	Enabled	Disabled	Disabled
Imager Power Model	Off	Off	Off	On	Off
Imager Data Model	Off	Off	On	On	Off
Mapping Power Model	Off	On	Off	Off	Off
Mapping Data Model	Off	On	Off	Off	Off
Transmitter Power Model	Off	Off	Off	Off	On
Transmitter Data Model	Off	Off	Off	Off	On

Several flight software tasks are implemented. These include a Sun-pointing task, Earth-pointing task, target-pointing task, map-pointing task, MRP control task, and a waypoint feedback control task. Depending on the flight mode, these tasks are turned on or off, primarily to determine which attitude reference should be used. A summary of each task's status in each flight mode is provided in Table 7.4. The sun-pointing, earth-pointing, target-pointing, and map-pointing tasks all use Basilisk's `locationPointing()` module and output an attitude guidance message which includes the MRP attitude error $\boldsymbol{\sigma}_{B/R}$. The attitude guidance message is ingested by the `mrpFeedback()` module, which outputs a commanded torque. This commanded torque is utilized by the `rwMotorTorque()` module to compute reaction wheel motor torques and send a motor command message to the three reaction wheel state effectors in the dynamics task.

The waypoint feedback control task utilizes a feedback control law to regulate the state of the spacecraft to the desired Hill frame waypoint. The feedback control law outputs a force command, which the `externalForceTorque()` dynamics module utilizes to pass the commanded force to the spacecraft.

In addition to several flight software tasks, a dynamics tasks is also implemented which holds the majority of the modules in the simulation. Gravity effectors for the asteroid, sun,

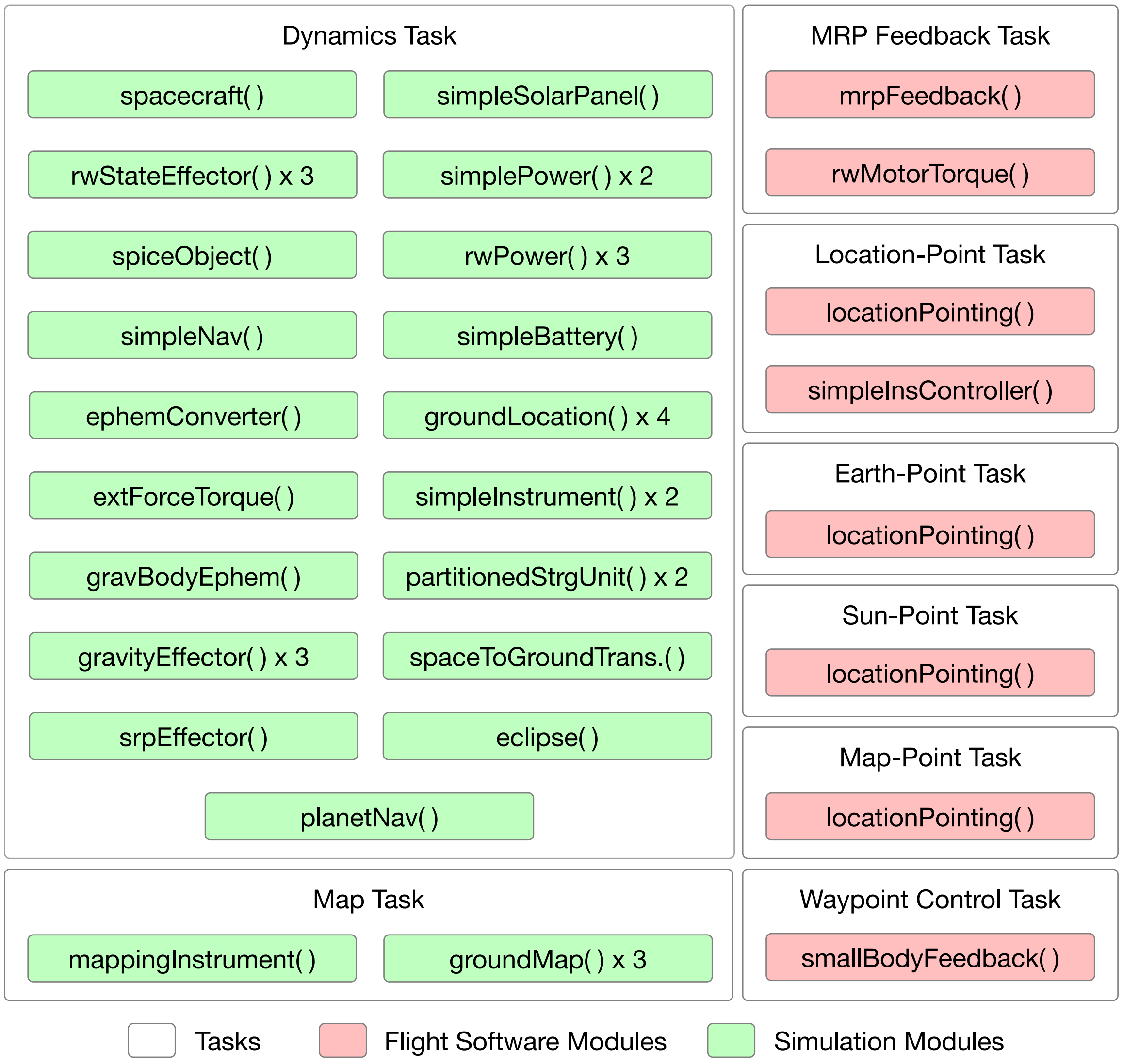


Figure 7.8: Basilisk simulation diagram.

and the Earth are implemented. A `planetNav()` module is also implemented for the asteroid, which creates an ephemeris message utilized by the relevant flight software modules. Likewise, a `simpleNav()` module performs the same function, but for the spacecraft state. The `planetNav()` and the `simpleNav()` modules can optionally add noise to the states to imitate a navigation system.

Several dynamics modules are connected to the spacecraft. As previously stated, the commanded force is passed to the spacecraft with the `extForceTorque()` module. Additionally, a `solarRadiationPressure()` module is implemented. A cannonball SRP module is utilized. Finally, each reaction wheel state effector is connected to the spacecraft for the purposes of attitude control. Lastly, the `eclipse()` module utilizes the state of the asteroid and the spacecraft to indicate whether or not the spacecraft is in eclipse.

A representative power system is modeled on-board the spacecraft. At the center of the power system is a `simpleBattery()` module. The battery receives power generation and consumption messages from each other power module to compute the storage level at each time step. Solar panels are modeled using the `simpleSolarPanel()` module, which computes power generation based on the area of the panels, the efficiency of the panels, and the solar incidence angle. Instrument and transmitter power models are also implemented with the `simplePowerSink()` module.

An on-board data system is also modeled. This system is modeled using two tasks - the dynamics task and the mapping task. The dynamics tasks is always on, but the mapping pass is disabled for all modes except for the mapping mode. This is done to minimize required computation. In the mapping task, three `groundMap()` modules are connected to a `mappingInstrument()`. The `groundMap()` module loops through each mapping point and checks for three things: a.) the spacecraft is within the elevation requirements of the point, b.) the point is within the instrument's field-of-view, and c.) the spacecraft is within the required azimuth angle band. A vector of access messages are then passed to the `mappingInstrument()`, which passes the data on to a `partitionedStorageUnit()`. This `partitionedStorageUnit()` in the mapping task keeps track of the points that have been imaged and those that have not. This serves a different function than the `partitionedStorageUnit()` in the dynamics task. In the dynamics task, two `simpleInstrument()` modules are implemented. One `simpleInstrument()` module is used in conjunction with the `simpleInstrumentController()` to image the ground targets if the imaging mode is entered. The other `simpleInstrument()` module is used keep track of the amount of data generated by mapping. This module provides a scalar value for data generated and does not keep track of the specific points. Both of these instruments pass the data to the `partitionedStorageUnit()` in the dynamics task.

Not shown in Figure 7.8 is the addition of a `smallBodyNavEKF` and an additional `simpleNav`

Table 7.5: Spacecraft parameters.

General Spacecraft Parameters	
Mass	330 kg
Dimensions	1.38 x 1.04 x 1.58 m
ΔV Limit	40 m/s
Power System	
Solar Panel Area	1.0 m^2
Solar Panel Efficiency	0.20
Instrument Power Draw	30 W
Transmitter Power Draw	15 W
Battery Capacity	100 Whr
Data & Communications System	
Data Buffer Storage Capacity	125 GB
Transmitter Baud Rate	120 Mbps
Instrument Baud Rate	8 Mbps
Map Instrument Baud Rate	8 Mbps
Initial Spacecraft State	
Radius	750 m
Azimuth Angle	$\mathcal{U}[30, 90, 150, 210, 270, 330]$ deg
Polar Angle	$\mathcal{U}[15, 45, 75, 105, 135, 165]$ deg

module. The `smallBodyNavEKF` is a Basilisk module that implements an extended Kalman filter for small body navigation. The `simpleNav` module is the Basilisk module that implements a second-order Gauss-Markov error model for translational navigation measurements. The `smallBodyNavEKF` is only utilized in some experiments.

7.5.2 Initial Conditions

The parameters of the spacecraft may be found in Table 7.5. The modeled spacecraft is the same small satellite used throughout this dissertation. These parameters are balanced to create a scenario in which the spacecraft must make tradeoffs between resource constraints, science collection, and downlink. The initial conditions for the asteroid orbit, size, and rotation may be found in Table 7.6. These parameters are based on those of Bennu [110, 111].

Table 7.6: Asteroid parameters.

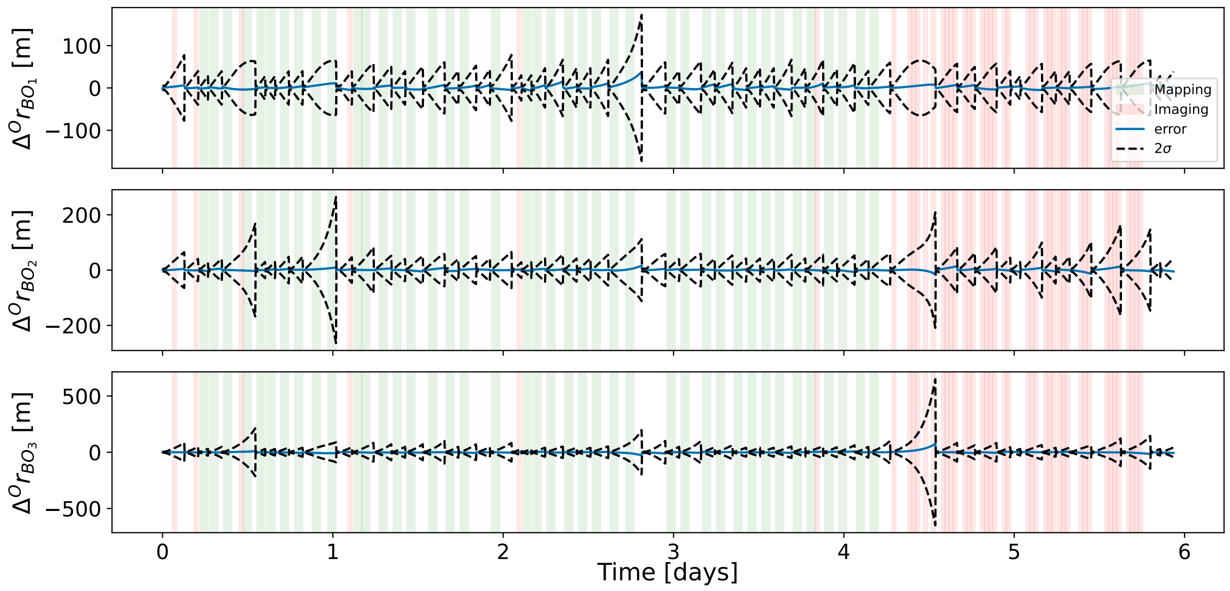
Orbital Parameters	
Semi-Major Axis, a	1.1259 AU
Eccentricity, e	0.016975
Inclination, i	0.0027666 deg
Long. of Ascend. Node, Ω	177.42 deg
Arg. of Periapsis, ω	284.26 deg
True Anomaly, f	357.30 deg
Size and Rotation	
Shape	Spherical
Rotation Period	4.297461 hr
Rotation Pole	Orbit Normal
Mean Radius	250 m
Gravitational Constant	4.892 m^3/s^2

7.6 Simulator Validation

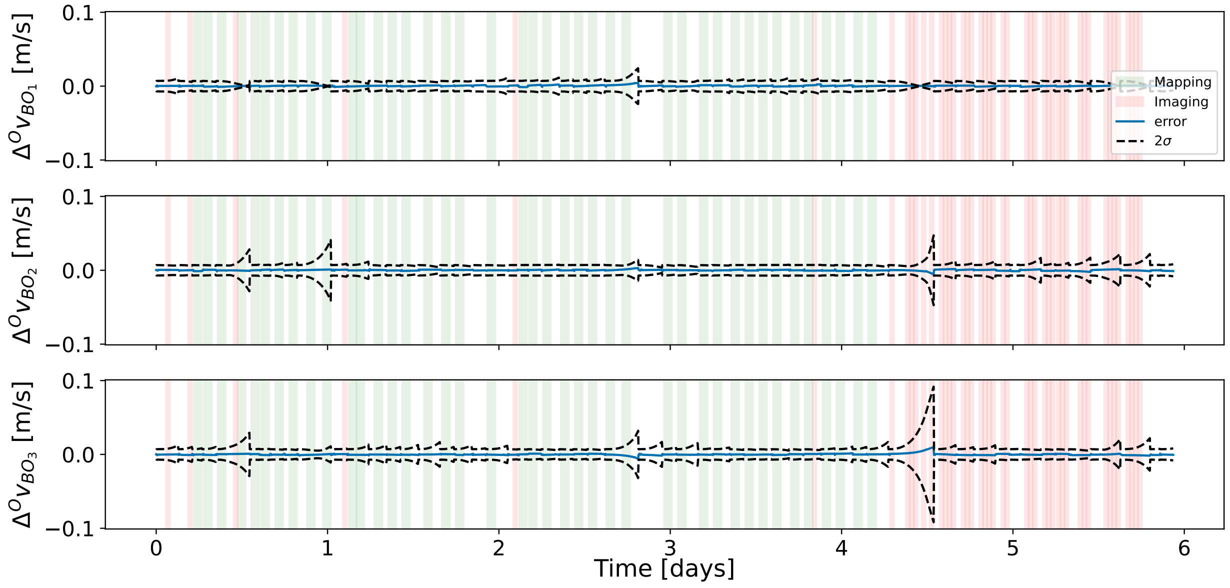
To validate the functionality of the simulator, this section uses a hand-crafted trajectory of actions designed for a single initial condition to ensure the mission is feasible and the simulator is functioning as expected. The initial condition is sampled from the aforementioned tables. In this example, the optional navigation mode is utilized. The hand-crafted trajectory of actions guides the spacecraft through the mapping waypoints, periodically performing science and resource management modes.

In Figure 7.9, the position and velocity errors of the EKF are presented. The covariance grows in between navigation modes when measurements are not being collected, but quickly snaps to a lower bound during the navigation modes. Navigation updates are typically performed immediately before mapping modes, which are shown in green. This is done to ensure that mapping will occur inside the designated region and that the spacecraft has not drifted outside of them due to navigation errors. Generally speaking, mapping is performed in the earlier phases of the mission, and ground target imaging is performed later. The ground target imaging mode is represented in red.

In addition to management of the estimation error, management of the buffer level, stored power, and ΔV is important. Plots of these resources are provided in Figure 7.10. The level of



(a) Position error and associated 2σ covariance bounds.



(b) Velocity error and associated 2σ covariance bounds.

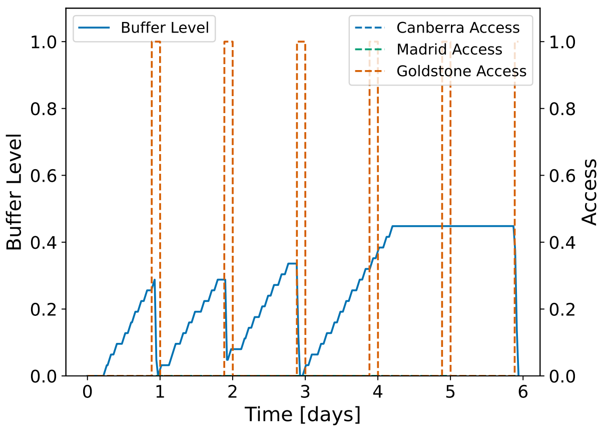
Figure 7.9: EKF position and velocity error over 6 days of operations. Mapping modes are shown in green. Imaging modes are shown in red.

the data buffer is provided in Figure 7.10a. Due to the 24-hour cadence of the DSN access, the Goldstone station is the only station available during the allotted communication window each day. The first three downlink opportunities are utilized, and the second two are skipped. The last downlink opportunity is utilized to downlink the rest of the mapping and imaging data. In Figure 7.10b, the stored power is provided. The power is managed such that the battery is not depleted fully. The charging modes are shown in yellow, demonstrating that the charging mode is working and the battery quickly reaches capacity during the charging mode. Finally, the total ΔV consumed is provided in Figure 7.10c. The maneuvers to new waypoints are shown in purple. Maneuvers to new waypoints consume fuel at the highest rate, but station keeping at the waypoints also contributes to the consumption. The ΔV is kept within the budget and the maneuvers are performed as necessary to reach the waypoints and facilitate science collection. This demonstrates that a nominal mission profile will not result in a complete depletion of fuel. However, unnecessary maneuvers and poor management of the state estimate can result in a violation of this resource limit.

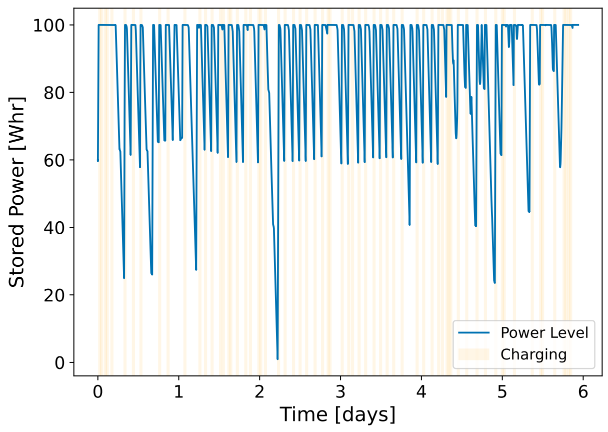
The trajectory of the spacecraft expressed in the sun-asteroid Hill frame is provided in Figure 7.11. The spacecraft traverses the entire range of polar angles, moving from azimuth angle to azimuth angle to perform mapping and imaging activities.

The trajectory of the spacecraft expressed in the body frame of the asteroid is displayed in Figure 7.12, along with the three maps. The green points represent mapping points collected. As evidenced by these plots, the prescribed actions manage to collect the vast majority of each map. Furthermore, every ground target is collected.

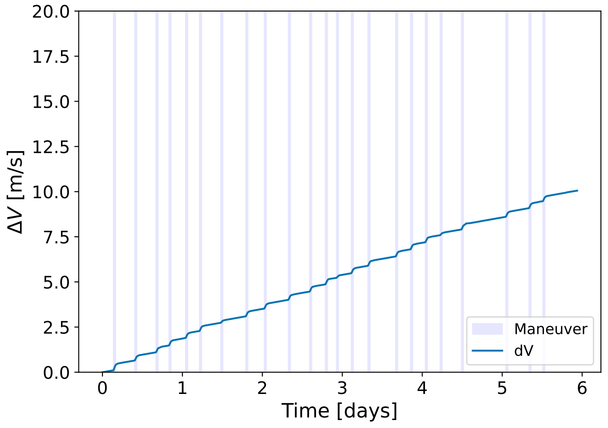
Finally, in Figure 7.13, the reward sum on a per-step basis is shown. Reward accumulation during the mapping modes occurs at a very steady rate, with the downlinks resulting in large increases in cumulative reward. During the imaging phase, the increase in overall reward occurs in larger increments because there are far less ground targets than mapping points. Finally, the large increase in reward at the very end of the simulation is due to the last downlink, which covers the entirety of a map and over half of the ground targets. This plot suggests that following a



(a) Data buffer level and DSN access.

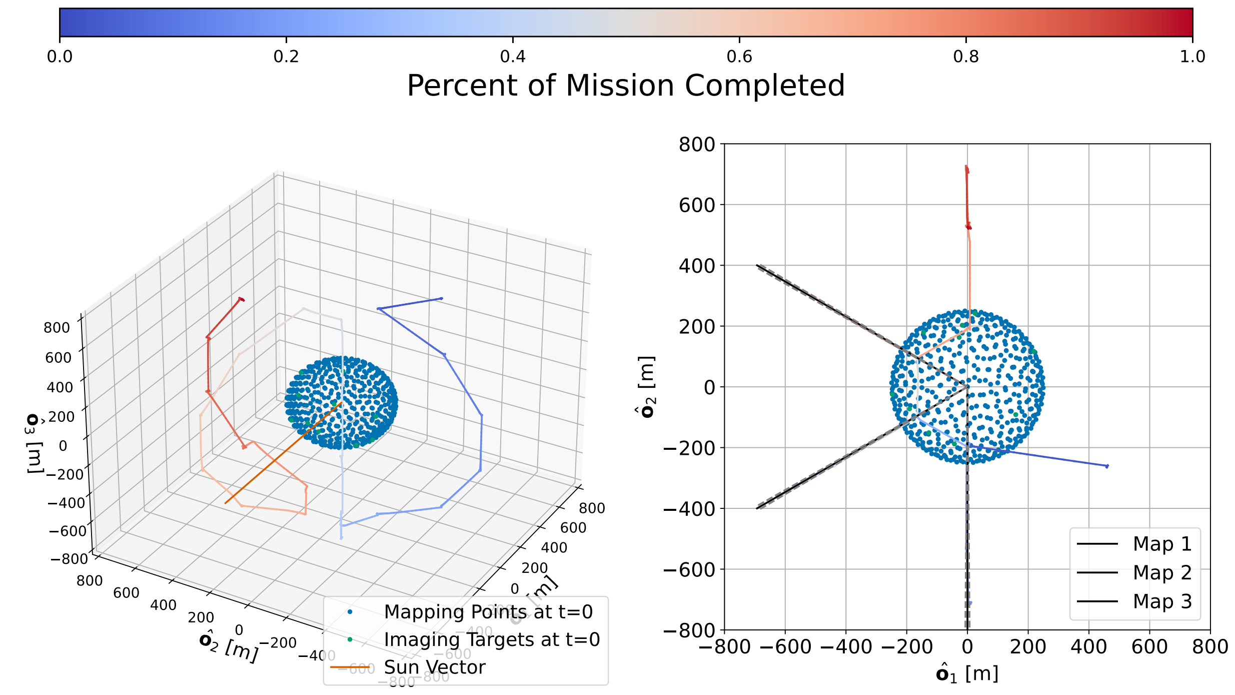


(b) Stored power with charging modes included.



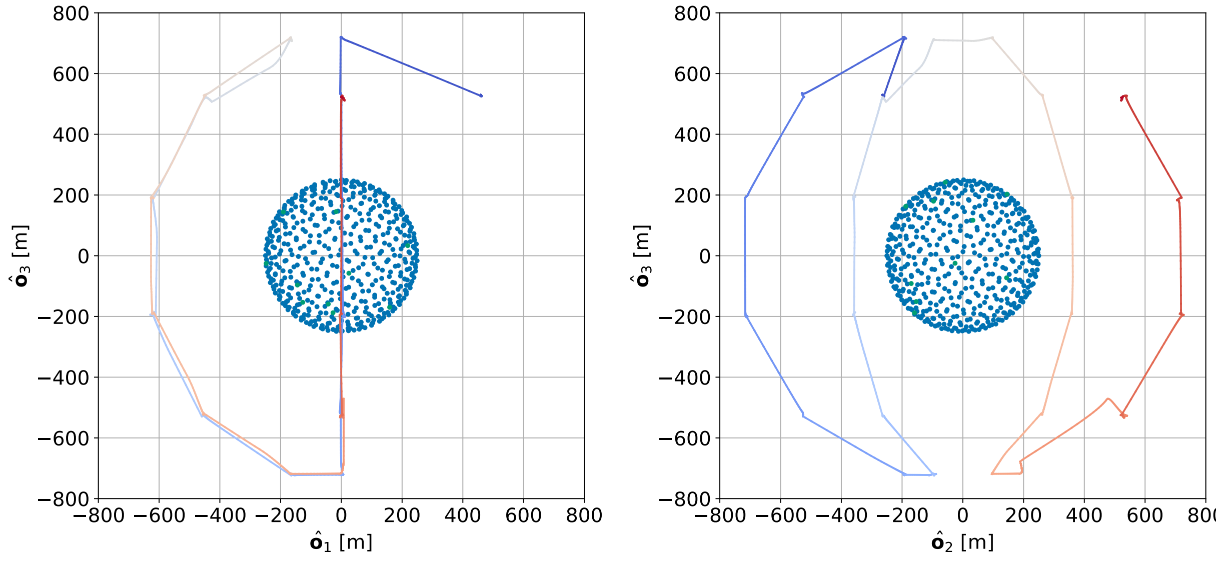
(c) Total ΔV consumed with maneuver modes included.

Figure 7.10: Spacecraft resources over time.



(a) 3D view.

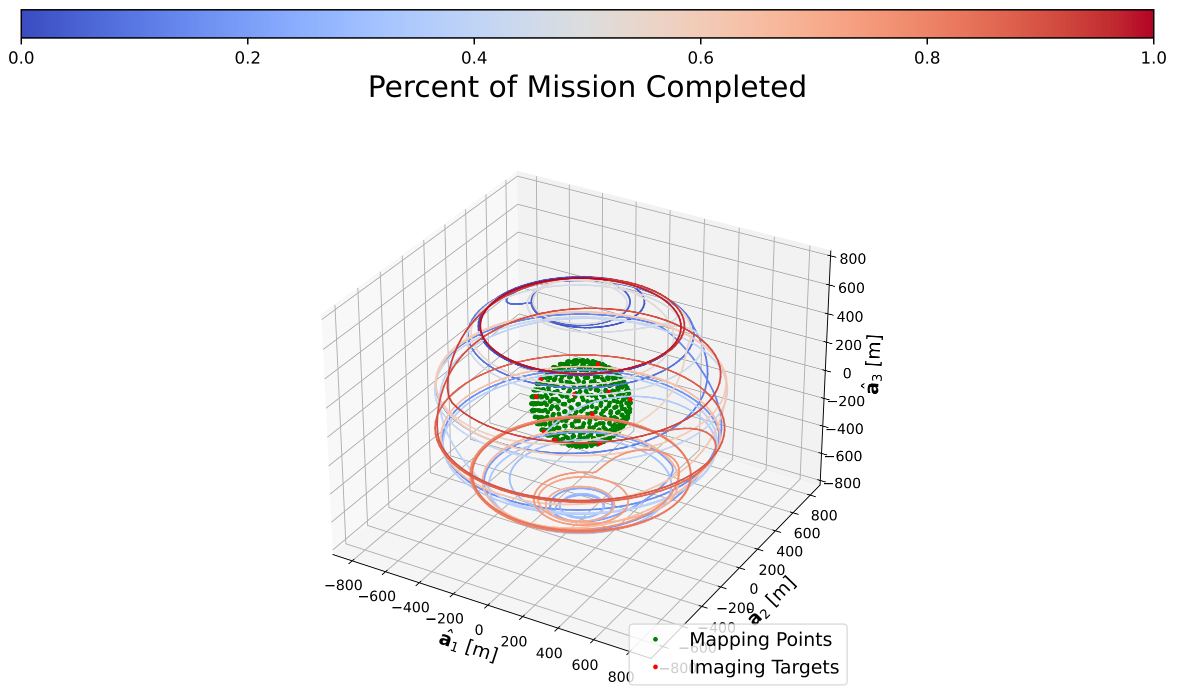
(b) $\hat{\mathbf{o}}_1$ & $\hat{\mathbf{o}}_2$ plane.



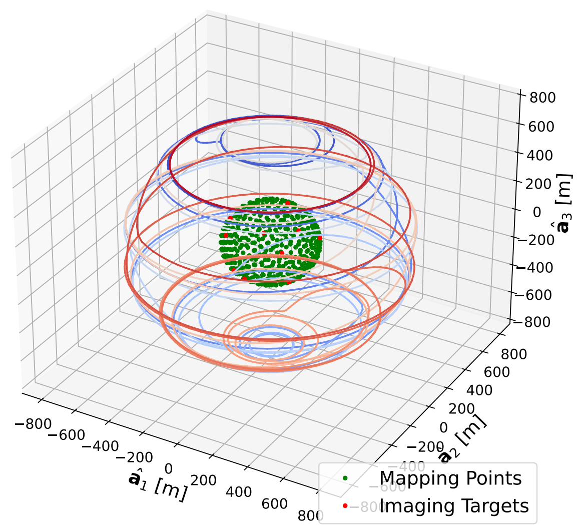
(c) $\hat{\mathbf{o}}_1$ & $\hat{\mathbf{o}}_3$ plane.

(d) $\hat{\mathbf{o}}_2$ & $\hat{\mathbf{o}}_3$ plane.

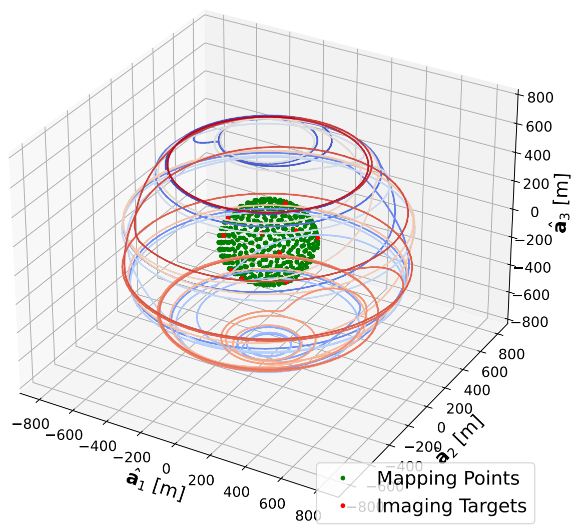
Figure 7.11: Trajectory of the spacecraft expressed in the sun-asteroid Hill frame following the prescribed actions.



(a) 6 PM LST map.



(b) 2 PM LST map.



(c) 10 AM LST map.

Figure 7.12: Trajectory of the spacecraft expressed in the asteroid body frame following the prescribed actions.

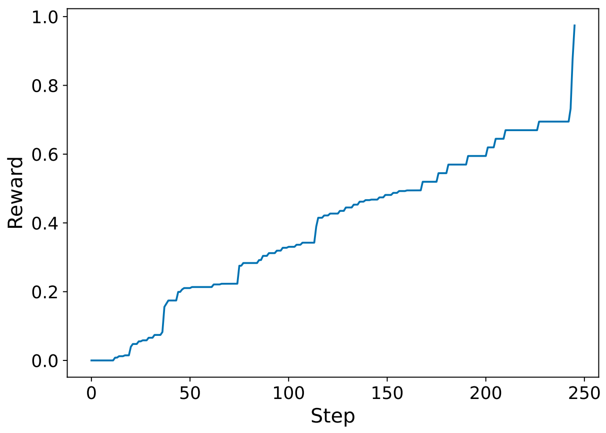


Figure 7.13: Cumulative reward each decision-making interval.

good trajectory of actions, the reward is not temporally sparse, which should help learning. The prescribed trajectory of actions results in 0.97 reward in total for this initial condition. Every ground target is collected, so the 0.03 in missing reward is due to a few map points not being collected and downlinked. It can be concluded from these results that a human expert can craft a trajectory of actions that results in almost all the possible reward being collected. However, if the initial conditions were to change, the entire trajectory of actions would need to be redesigned, which takes several hours by hand. This is the motivation for using RL to solve this problem.

7.7 Conclusion

This chapter provides a detailed description of the small body science operations problem that adequately addresses resource constraints, GNC activities, and their relationship to science data. The problem is formulated as a Markov decision process, which includes the definition of the state space, action space, reward function, and transition function. The parameters of the high-fidelity astrodynamics simulations were also presented. A hand-crafted solution to the problem was developed to validate the simulator and show the delicate balance between managing resources and collecting and downlinking science data. This problem formulation and simulator will be solved in the next chapter using some of the RL techniques discussed in this dissertation.

Chapter 8

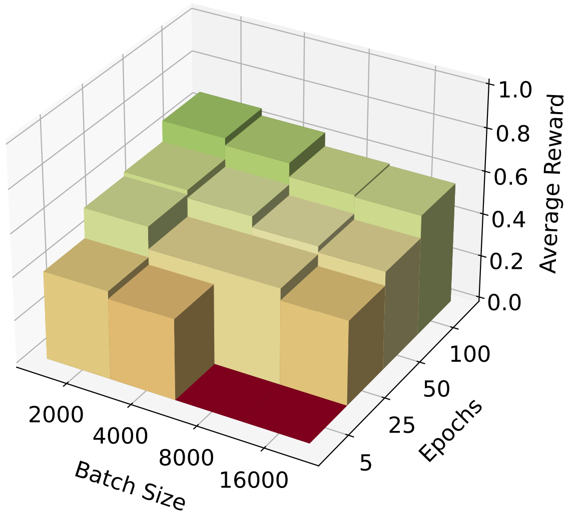
Small Body Science Scheduling Results

8.1 Introduction

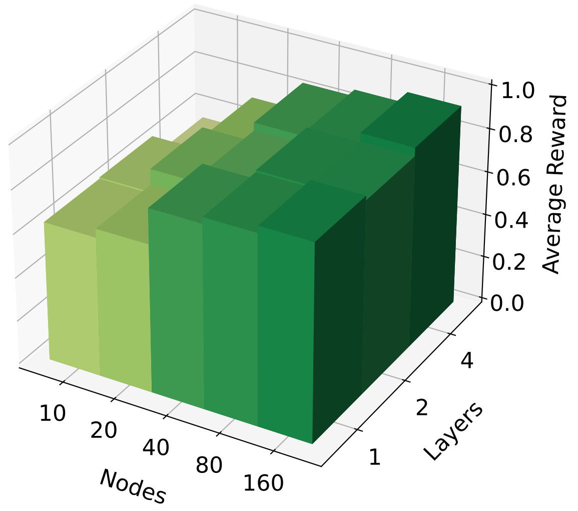
This chapter presents several experiments for the small body science scheduling problem presented in Chapter 7. Due to its good convergence properties and performance, PPO is selected as the algorithm of choice to solve and parameterize the small body science scheduling problem. The first experiment performed is a hyperparameter search over the batch size, number of epochs, network widths, and network depths. This experiment is identical to the PPO experiments performed in Chapter 5. After the hyperparameters are optimized, a search over the reward function in Equation 7.17 is performed to determine how image and mapping collection and downlink, as well as the failure penalty, should be weighted. Some of these trained policies are also deployed on the various observation types to determine the impact of the observation type on the performance of the policy. The policies trained with the truth state are deployed using noisy measurements of the state and the EKF belief state to determine the impact of noise and state estimation error on the performance of the policy. Finally, a set of policies are trained with the dedicated navigation mode and associated state return (i.e. covariances are added to the state return).

8.2 Hyperparameter Searches

To determine which hyperparameters should be selected for future experiments, two hyperparameter searches are performed for the small body science scheduling problem. For both experiments, reward is split evenly between collection and downlink for imaging and mapping, which are



(a) Hyperparameter optimization over batch size and epochs. Hyperparameters: width = 20, depth = 4, LR = 3e-5.



(b) Hyperparameter optimization over policy width and depth. Hyperparameters: batch size = 2000, epochs = 100, LR = 3e-5.

Figure 8.1: PPO hyperparameter searches for the small body science environment. Each bar represents the average of 5 trials.

also split evenly. The constants of the reward function in Equation 7.17 are $A = B = C = D = 0.25$, and the failure penalty is set to $F = 1$. In the first hyperparameter search, the batch size and number of epochs are varied. The network width and depth are fixed at 20 and 4, respectively. The learning rate is fixed at 3e-5. The results of this hyperparameter search are shown in Figure 8.1a. The best performing policy is trained with a batch size of 2000 and 100 epochs. This matches the results in Chapter 5, which found that smaller batch sizes and more epochs are beneficial. The second hyperparameter search varies the network width and depth. The batch size and number of epochs are fixed at 2000 and 100, respectively. The learning rate is fixed at 3e-5. The results of this hyperparameter search are shown in Figure 8.1b. The best performing policy is trained with a network width of 160 and a network depth of 1.

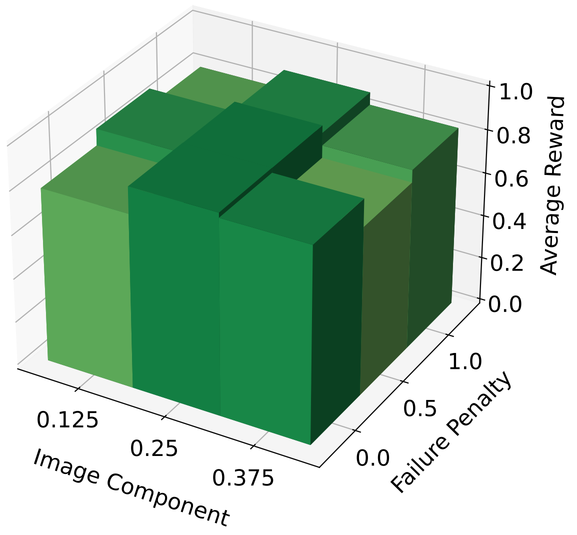


Figure 8.2: Average reward across the reward function parameters. Hyperparameters: batch size = 2000, epochs = 100, LR = 3e-5, width = 320, depth = 1.

8.3 Reward Function Engineering

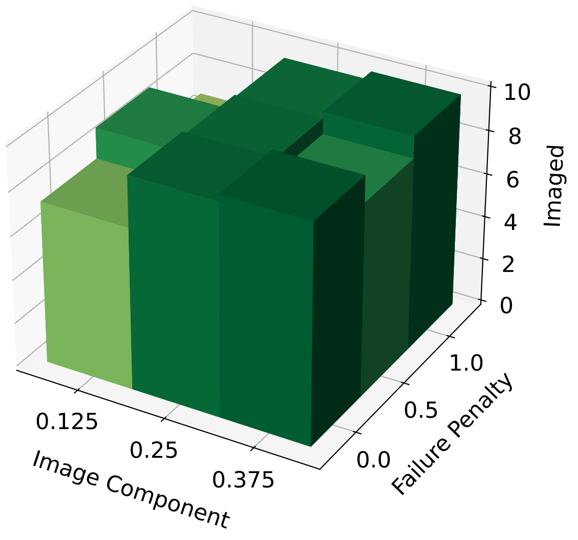
After the hyperparameters are tuned, a search over the reward function parameters in Equation 7.17 is performed using the optimized hyperparameters (the network width is increased to 320, however). The reward is equally split between collection and downlink, but the weighting towards imaging and mapping is varied from 25% to 75%. The failure penalty is also varied from 0 up to a penalty of $F = 1$. The average reward across these hyperparameters after training is provided in Figure 8.2. An imaging component of 0.125 means that $A = C = 0.125$, so the mapping components are $B = D = 0.375$. In terms of average reward, changing the relative weighting of imaging and mapping does not appear to have a large impact on the final average reward. However, because the failure penalty changes the range of reward, more metrics need to be collected to determine whether or not this is true.

In Figure 8.3, the average number of imaged and downlinked surface targets and collected and downlinked maps are presented. A clear dependency on the relative weight of surface target imaging vs. downlink is present in the results. When surface target imaging is weighted more, the average number of surface imaging targets collected and downlinked increases. When mapping is weighted more, the average number of mapping points collected and downlinked increases. Equal weighting of surface target imaging and mapping results in decision-making agents that collect and downlink a high amount of both surface targets and mapping points. This result makes intuitive sense as both types of science data are prioritized equally. In general, the failure penalty does not appear to have a large impact on these numbers. If the decision-making agent fails, it is penalized by not being able to collect and downlink further science data. This matches the results on the reward tuning experiments performed in Chapter 5 for the SSAEO scheduling problem.

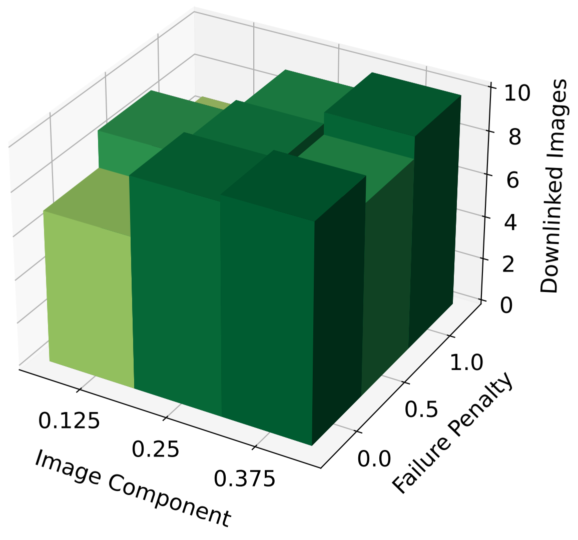
In Figure 8.4, the average length of the simulation and ΔV are provided for the reward function parameters. The average length of simulation is relatively constant over the reward parameters. However, a failure constant of 1 does result in the most stable average simulation length, which means the decision-making agent eliminates failures altogether. The ΔV is more variable, with no obvious dependency on the reward parameters, but all decision-making agents keep the ΔV well within the requirement of 40 m/s.

8.4 Policy Evaluation

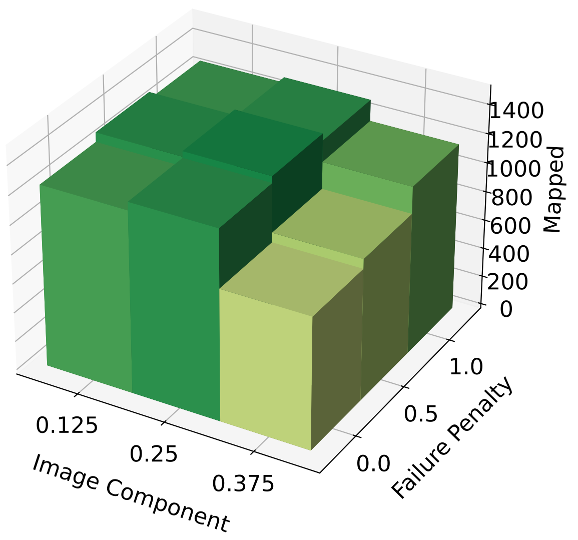
This section evaluates the trained policies using the following reward components: $A = B = C = D = 0.25$, $F = 1$. The science components of the reward function are equally weighted, and the failure penalty is set to the highest value. The first experiment performed in this section evaluates the policy trained with these reward parameters under various observation types. The nominal observation type observes the relative spacecraft state directly. The Basilisk `simpleNav` module observation type observes noisy measurements of the relative spacecraft state. If this observation type is utilized, a dead band equal to the 1σ values in Table 7.2 is added to the control law to ensure extra fuel is not spent trying to correct for measurement noise. The EKF observation type observes the EKF belief state, where the EKF continually ingests measurements to improve the state estimate. No controller dead band is required here. The results of these experiments for $N = 20$ trials is presented in Table 8.1. The nominal state observations result in the highest



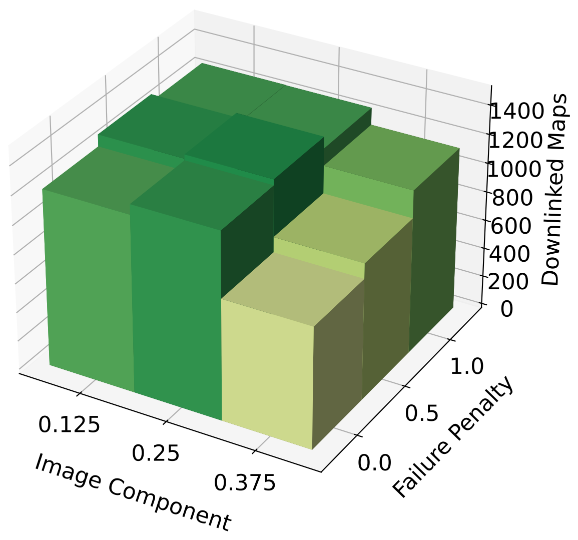
(a) Average number of collected images.



(b) Average number of downlinked images.



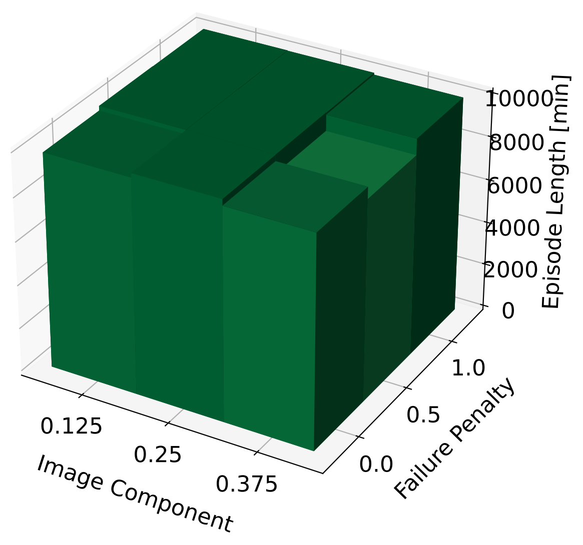
(c) Average number of collected mapping points.



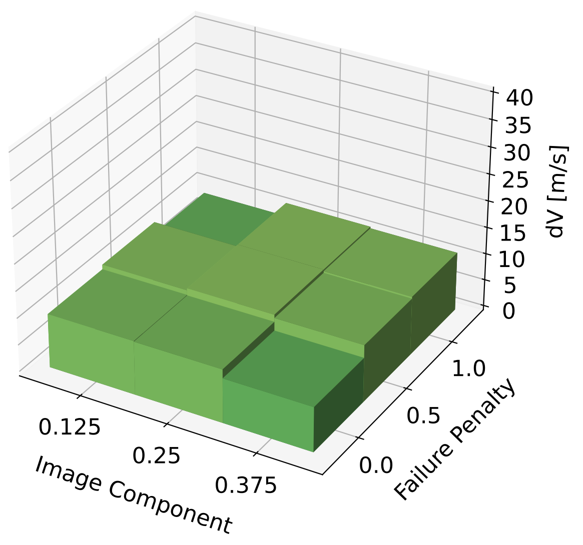
(d) Average number of downlinked mapping points.

Figure 8.3: Average number of landing sites and map points collected and downlinked over the reward components.

average reward, but the confidence intervals for each metric and each observation type overlap. There is no appreciable difference in performance between each observation type. This is a major feature of the problem formulation and training methodology, as a small amount of noise added to



(a) Average length of simulation.



(b) Average ΔV.

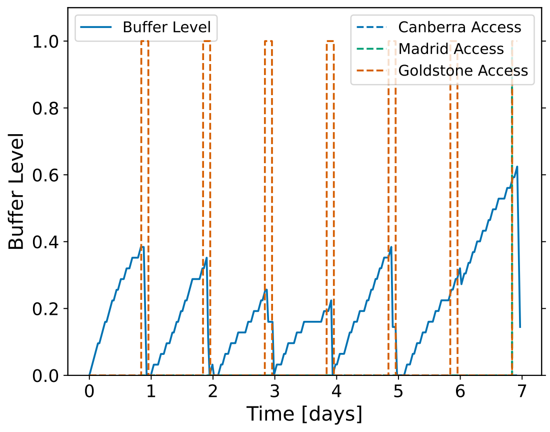
Figure 8.4: Average simulation length and ΔV across the reward function parameters.

Table 8.1: Trained policy deployed with different observation types. 7 day long planning horizon.

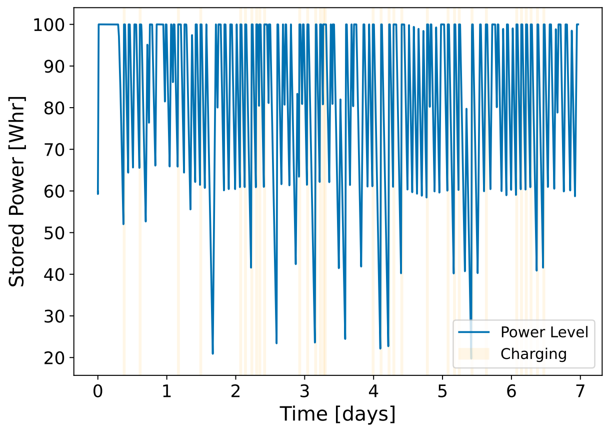
Metric	Nominal	SimpleNav	EKF
Average Reward	0.91 ± 0.02	0.88 ± 0.04	0.86 ± 0.05
Average ΔV	9.4 ± 0.3	9.5 ± 0.4	9.4 ± 0.3
Collected Images	9.5 ± 0.39	9.1 ± 0.5	8.5 ± 0.7
Downlinked Images	9.4 ± 0.39	9.0 ± 0.5	8.5 ± 0.7
Collected Map (6 PM LST)	451 ± 14	467 ± 9	468 ± 12
Collected Map (2 PM LST)	426 ± 30	390 ± 46	392 ± 47
Collected Map (10 AM LST)	443 ± 18	430 ± 30	455 ± 20
Downlinked Map (6 PM LST)	451 ± 14	467 ± 9	468 ± 12
Downlinked Map (2 PM LST)	408 ± 32	387 ± 45	381 ± 47
Downlinked Map (10 AM LST)	439 ± 18	424 ± 29	454 ± 20

the state does not significantly impact the performance of the decision-making agent.

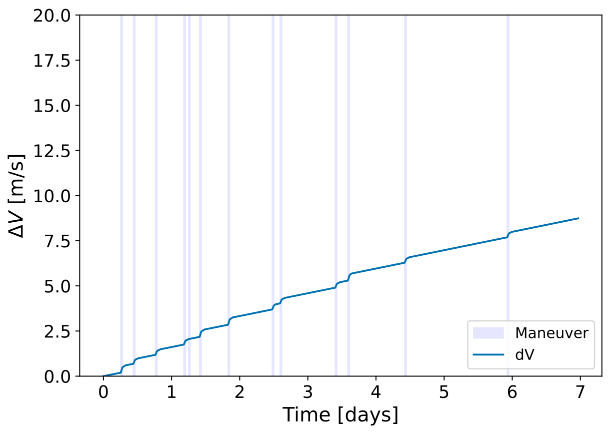
Data is collected on the policy utilizing the nominal observation type to compare it to the hand-crafted trajectory of actions in Chapter 7. The data buffer level, stored power, and ΔV are presented in Figure 8.5. The policy takes advantage of almost every downlink opportunity,



(a) Data buffer level and DSN access.



(b) Stored power with charging modes included.



(c) Total ΔV consumed with maneuver modes included.

Figure 8.5: Spacecraft resources over time.

keeping the data buffer well within the limits. The stored power also stays above 20 Whr at all times. Finally, the trained policy only performs 13 maneuvers in comparison to the 23 maneuvers performed by the hand-crafted trajectory.

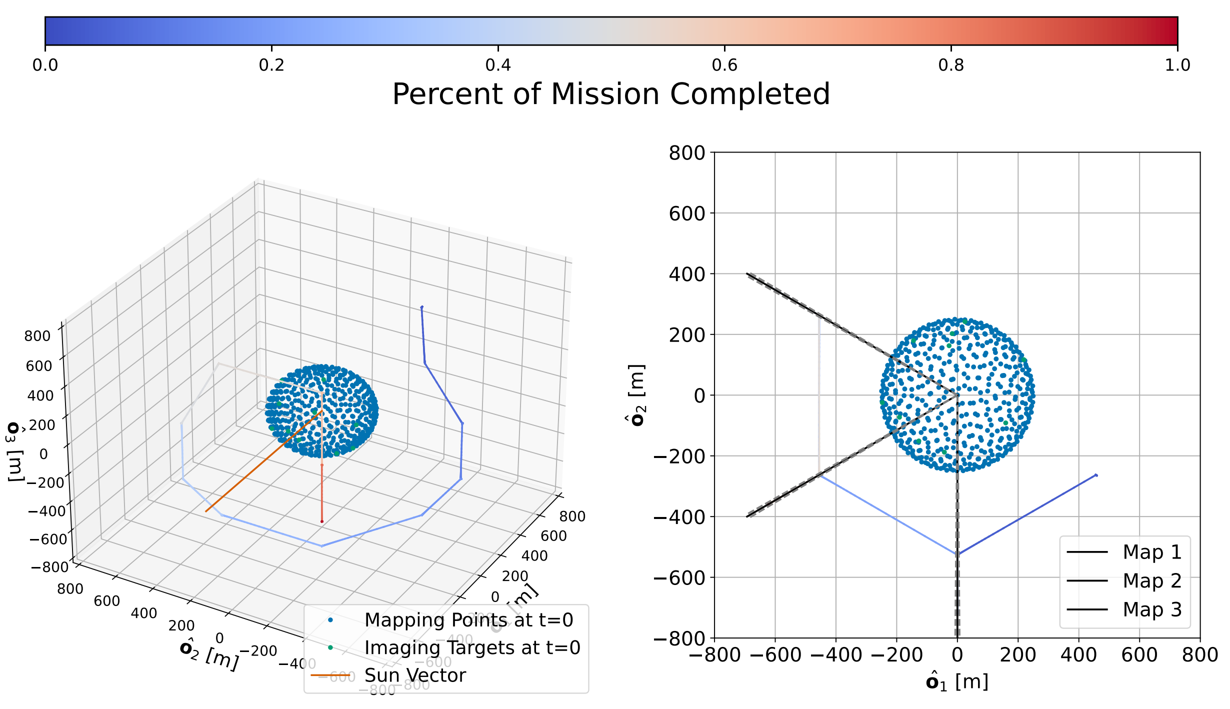
As evidenced by Figure 8.6, the reason that the decision-making agent performs fewer maneuvers is because it typically does not visit the northern and southern-most polar angles. The decision-making agent has learned that it can complete the majority of the mission without visiting these waypoints. However, the question of whether or not this impacts the number of maps collected remains. In Figure 8.7, the collected map points are displayed, as well as the trajectory of the spacecraft in the asteroid body frame. The number of total mapping points collected is almost identical to the number collected with the hand-crafted trajectory of actions. The decision-making agent has learned that visiting the extreme northern and southern polar angles is not necessary to complete the mission.

Finally, in Figure 8.8, the reward on a per-step basis is displayed. The reward plateaus at 0.96 at about 150 steps through the environment. While this may seem impressive in comparison to the hand-crafted trajectory of actions, the fact that the hand-crafted trajectory also considers the optional navigation mode must be considered. The performance of the decision-making agent with the navigation mode included will be investigated in the next section.

8.4.1 DSN Outage Experiment

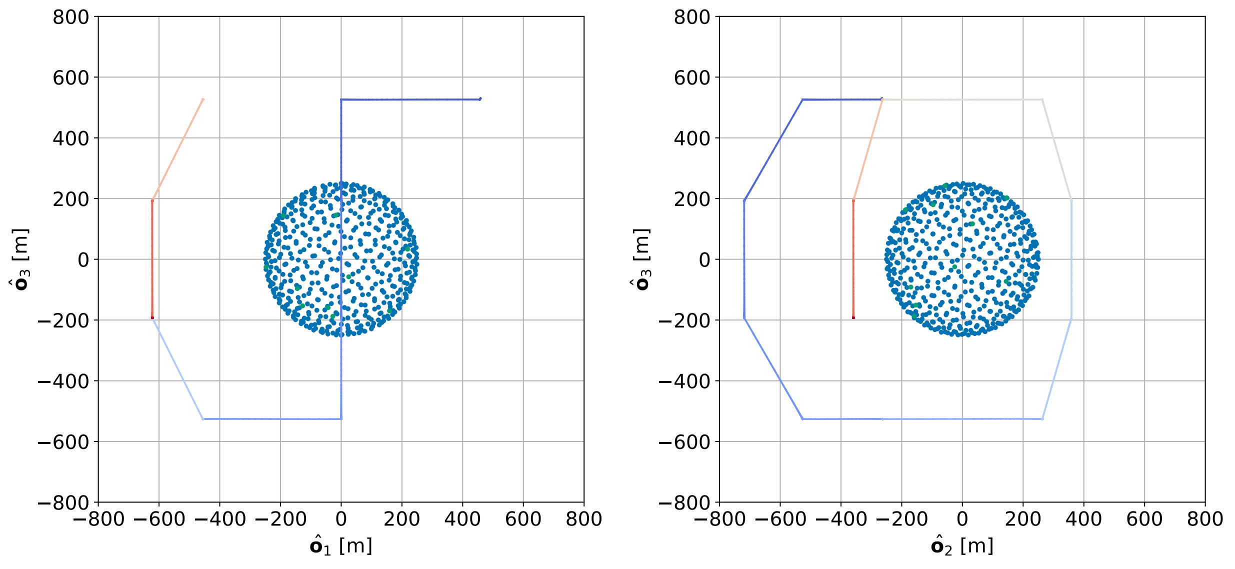
A large benefit of using reinforcement learning for planning and scheduling is that it yields closed-loop planning solutions as opposed to open-loop solutions, which allows the decision-making agent to respond to opportunistic science events or ground station outages. While uncommon, DSN outages can occur and place mission timelines in jeopardy. On October 11th, 2019, a DSN outage occurred at the Madrid station before a "late update" (an update to the spacecraft's trajectory) for the OSIRIS-REx mission [112]. Engineers had to scramble and compress this update within a four-hour window. In addition to trajectory and navigation updates, missed downlink windows can also have implications on future collection and downlink of science data. In this experiment, a DSN outage is simulated by simply removing the third downlink opportunity. The nominal buffer level and access times, without this removal, are shown in Figure 8.9a. The average policy outputs during the third downlink window are shown in Figure 8.9b. The highest probability action is action 10, which is downlink.

In Figure 8.10, the buffer level and policy outputs during the removed downlink window are shown. During the removed downlink window, the downlink state is replaced with 0. The decision-making agent responds by continuing the mapping and imaging campaign, requiring no input from the ground other than notification that the window has been removed. The downlink action goes



(a) 3D view.

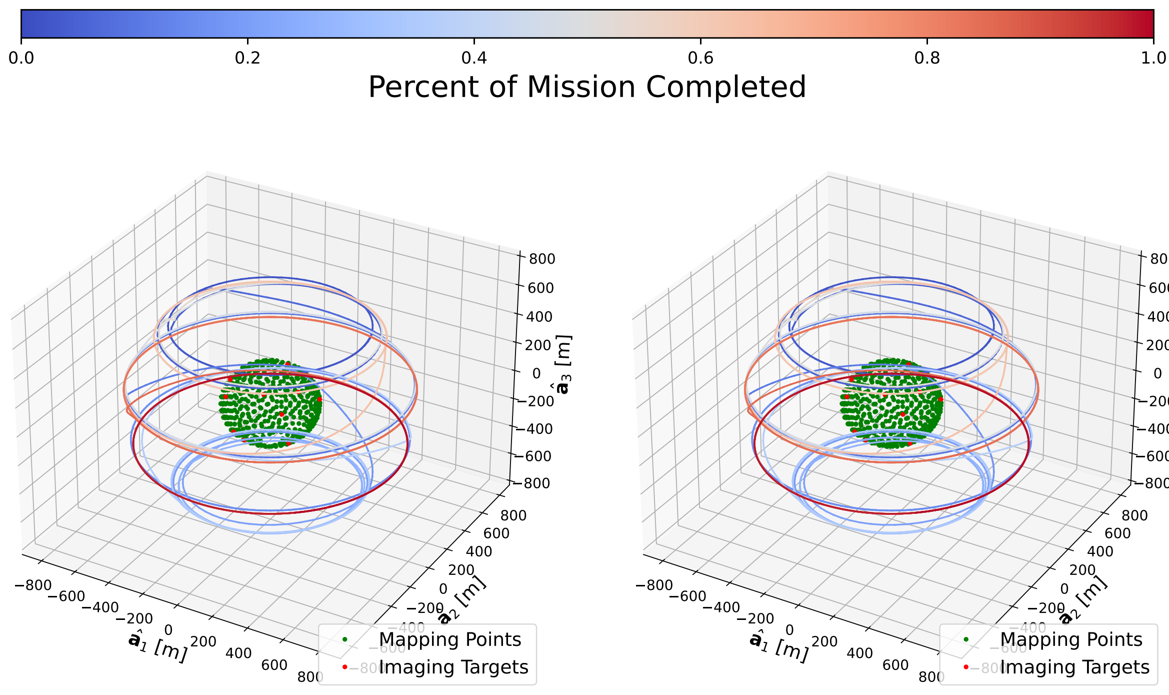
(b) $\hat{\mathbf{o}}_1$ & $\hat{\mathbf{o}}_2$ plane.



(c) $\hat{\mathbf{o}}_1$ & $\hat{\mathbf{o}}_3$ plane.

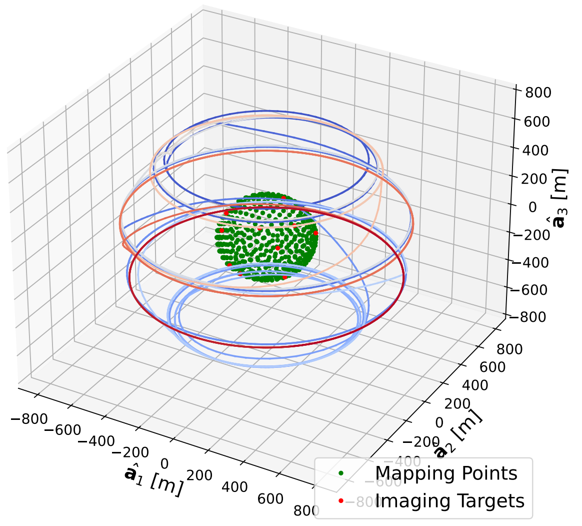
(d) $\hat{\mathbf{o}}_2$ & $\hat{\mathbf{o}}_3$ plane.

Figure 8.6: Hill frame trajectory of the spacecraft.



(a) 6 PM LST map.

(b) 2 PM LST map.



(c) 10 AM LST map.

Figure 8.7: Trajectory of the spacecraft in the asteroid body frame. Green points represent mapping points collected by the spacecraft.

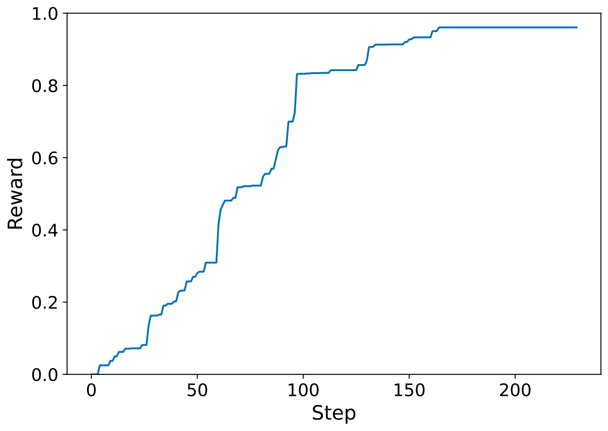
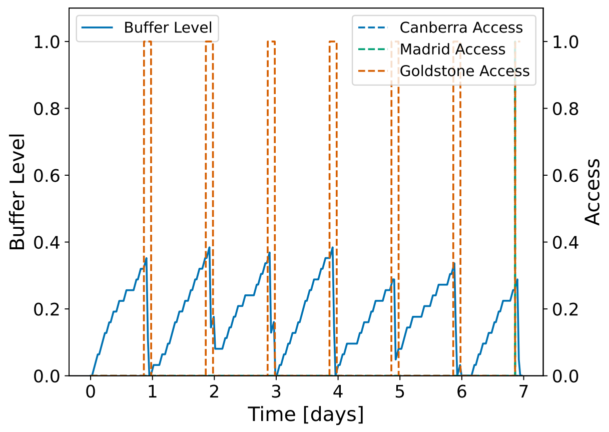
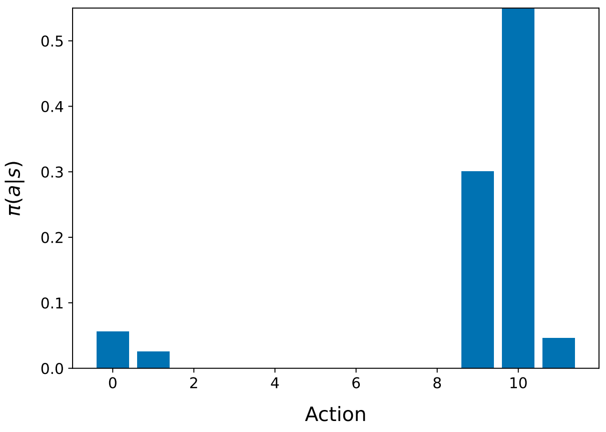


Figure 8.8: Cumulative reward each decision-making interval.



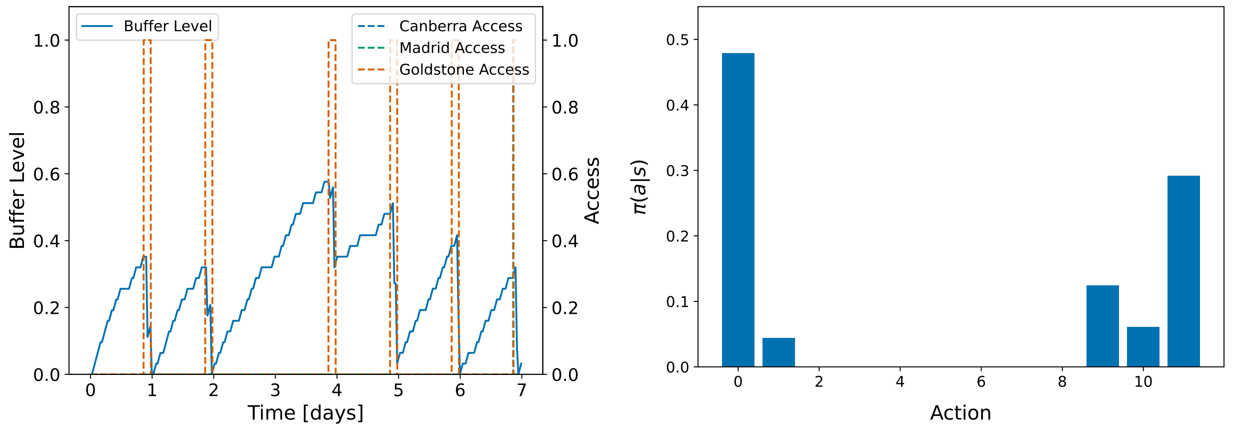
(a) Buffer level and DSN access.



(b) Policy outputs during third downlink window.

Figure 8.9: Buffer level and policy outputs during nominal DSN schedule. Action 0: Charge. Actions 1-8: Waypoint Maneuver. Action 9: Map. Action 10: Downlink. Action 11: Image Surface Target.

from the highest probability action to the one of the lowest probability actions. This is a major benefit of using reinforcement learning for planning and scheduling, as the decision-making agent can respond to changes in the environment with little or no human intervention. No expensive re-planning efforts on board the spacecraft are required either. The entire trajectory of actions does not need to be re-optimized using an integer program. Furthermore, this also highlights the utility of using this representation of the downlink state. In the MDP formulation of the agile EOS scheduling problem, the ECEF position is relied on to help the agent understand when downlink



(a) Buffer level and DSN access. (b) Policy outputs during removed downlink window.

Figure 8.10: Buffer level and policy outputs during disrupted DSN schedule. Action 0: Charge. Actions 1-8: Waypoint Maneuver. Action 9: Map. Action 10: Downlink. Action 11: Image Surface Target.

opportunities occur. The benefit of doing this is that it doesn't require a pre-computation of ground station access. However, relying on ECEF positions for ground station access doesn't easily allow for the removal of a downlink window. Therefore, a temporal representation for upcoming downlink windows should be utilized in future work.

8.5 Navigation Mode

8.5.1 Policy Evaluation

In Table 8.2, the policy trained with the dedicated navigation mode is benchmarked over $N = 20$ 7-day planning horizons. When compared to Table 8.1, it's evident that PPO trained with the navigation mode is capable of producing policies equivalent (in terms of average reward) to those trained without the navigation mode and associated state uncertainty. This is an impressive result, demonstrating that PPO can not only manage spacecraft resources, but state uncertainty as well, simply by monitoring the diagonals of the error covariance matrix. If autonomous guidance and relative navigation capabilities mature enough for adoption, reinforcement learning is a viable method for on-board planning and scheduling of small body science missions. Future work should

Table 8.2: Benchmarking the policy trained for the MDP with the dedicated navigation mode. 7 day long planning horizon.

Metric	Value
Average Reward	0.91 ± 0.02
Average ΔV	11.9 ± 0.5
Collected Images	9.7 ± 0.3
Downlinked Images	9.4 ± 0.4
Collected Map (6 PM LST)	455 ± 12
Collected Map (2 PM LST)	452 ± 20
Collected Map (10 AM LST)	400 ± 17
Downlinked Map (6 PM LST)	441 ± 20
Downlinked Map (2 PM LST)	448 ± 20
Downlinked Map (10 AM LST)	388 ± 23

investigate how state-of-the-art autonomous guidance and relative navigation methods impact the performance of the decision-making agent and how the problem formulation may need to be adjusted to account for these impacts.

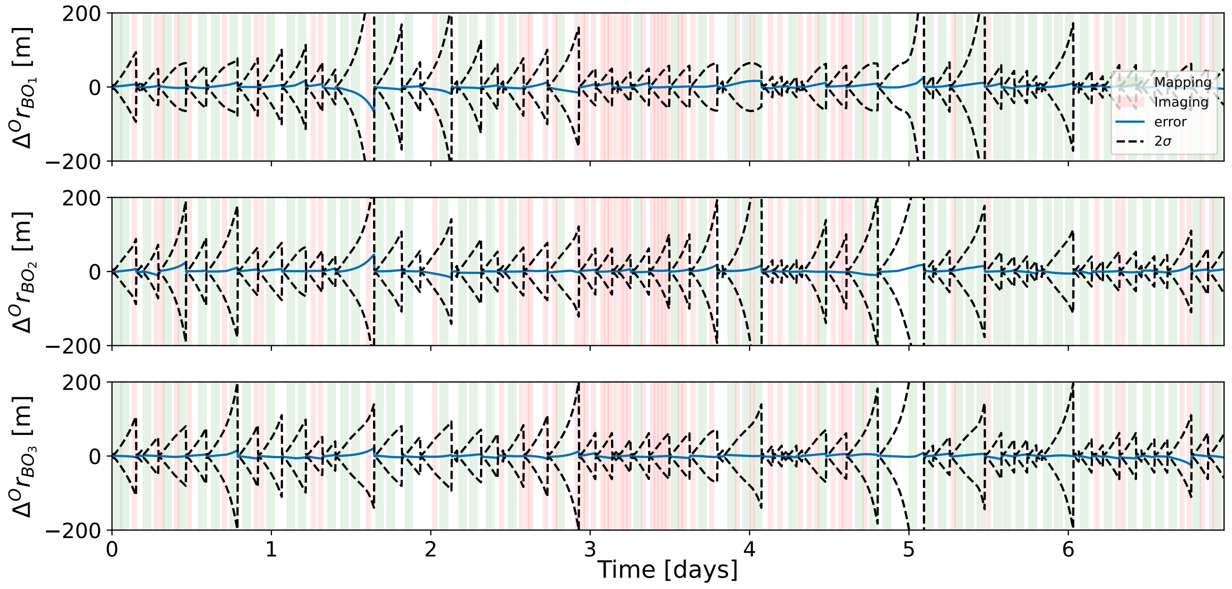
To compare to the hand-crafted trajectory of actions in Chapter 7, the same data is collected for a single run of the policy and evaluated. The position and velocity state error, along with the 2σ covariance bounds, are displayed in Figure 8.11. The decision-making agent periodically updates its state estimate with new measurements, reducing the state error covariance such that mapping can be conducted. The data buffer level, stored power, and ΔV are presented in Figure 8.12. The policy takes advantage of almost every downlink opportunity, keeping the data buffer well within the limits. The stored power also always stays above 20 Whr. Finally, the trained policy only performs 19 maneuvers in comparison to the 23 maneuvers performed by the hand-crafted trajectory. The decision-making agent attempts to make more maneuvers, but these maneuvers are not actually performed by the spacecraft because the required amount of time before the next maneuver can be taken has not passed. The trajectory of the spacecraft in the sun-asteroid Hill frame is displayed in Figure 8.13. The decision-making agent learns to make multiple passes through the mapping waypoints. This is an unintuitive result, but the decision-making agent has learned to map and then move to the next waypoint instead of waiting for the other side of the body to become visible. The trajectory of the spacecraft in the asteroid body frame is displayed in Figure 8.14. As evidenced by these plots, the decision-making agent collects the majority of mapping points.

8.5.2 Management of the State Estimate

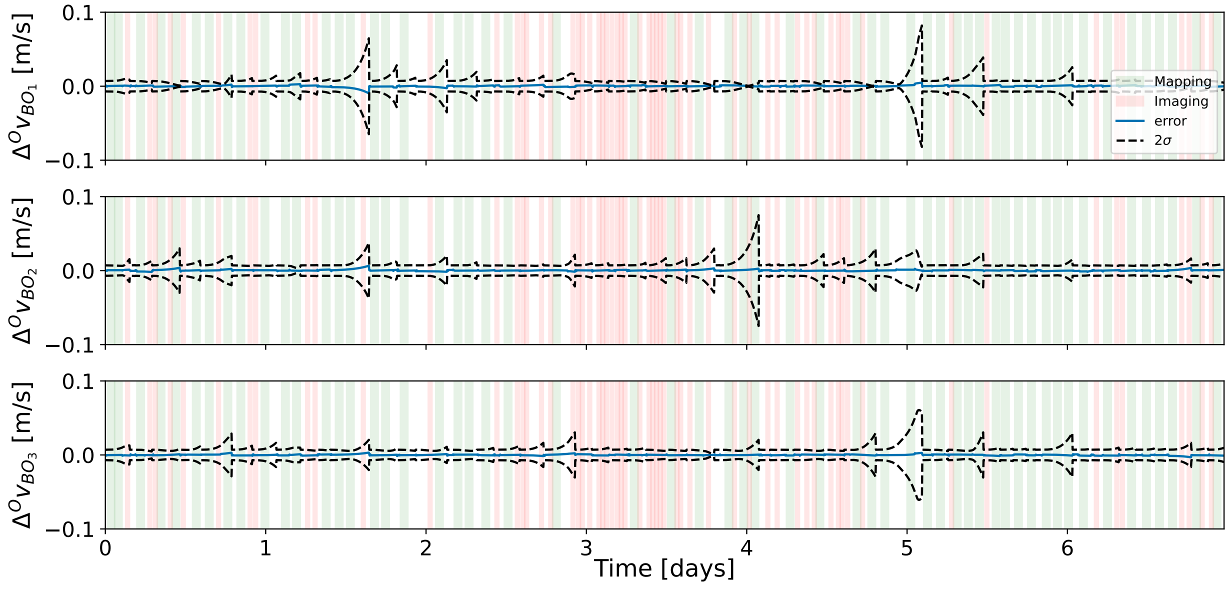
Based on Figure 8.11a, it appears that the decision-making agent has learned to manage its state estimate. However, it's unclear if the agent has learned to assign fixed probability to the navigation update mode or if selection of the navigation update mode is dependent on the value of the state-error covariance. To determine this, the policy is run in the environment and, the covariance matrix is collected, and the policy distribution is evaluated for different values of the diagonal of the covariance matrix. These results are presented in Figure 8.15. For $10^1 \leq \|\text{diag}(P)\| < 10^2$, when the covariance and state error are the lowest, the lowest probability action (outside of the maneuvers) is the navigation update. As the state error covariance grows, the navigation update becomes the most likely action, as evidenced by Figure 8.15d where $\pi(\text{nav update}|s) \approx 0.55$. These results suggest that the decision-making agent has learned to manage its state estimate based on the observations provided.

8.6 Conclusion

This chapter explores the application of PPO to the small body science operations problem. A hyperparameter search over the PPO hyperparameters and reward function components is performed. Small batch sizes and larger networks are shown to produce the best performing policies. In terms of the reward function experiments, equal weighting between surface target imaging and mapping produces the best policies. Finally, the performance of the trained policies is evaluated using direct observations of the state, noisy observations of the state, and filtered observations produced by an EKF. The decision-making agent is shown to be robust to noise in the state observations, even when only trained using direct observations of the state. The decision-making agent is also shown to be capable of responding to changes in the environment, such as a DSN outage,

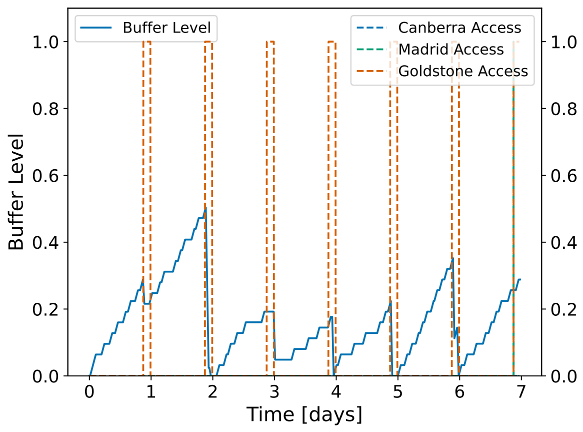


(a) Position error and associated 2σ covariance bounds.

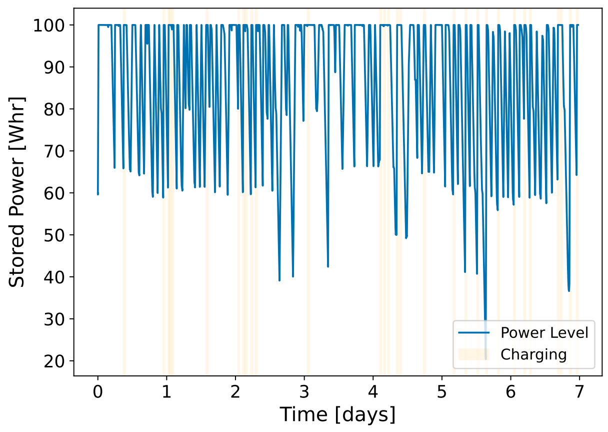


(b) Velocity error and associated 2σ covariance bounds.

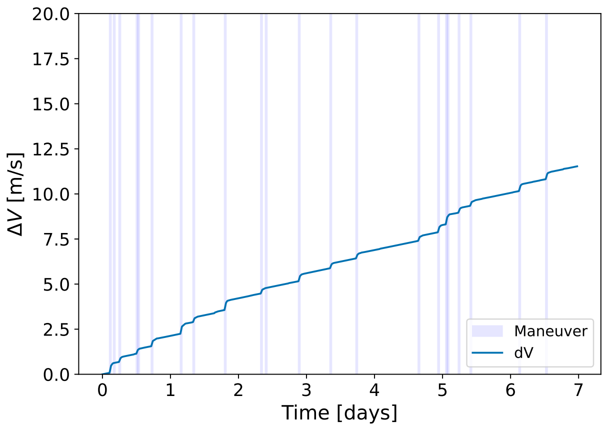
Figure 8.11: EKF position and velocity error over 7 days of operations utilizing the optional navigation mode. Mapping modes are shown in green. Imaging modes are shown in red.



(a) Data buffer level and DSN access.

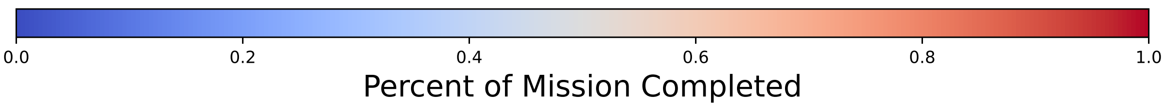


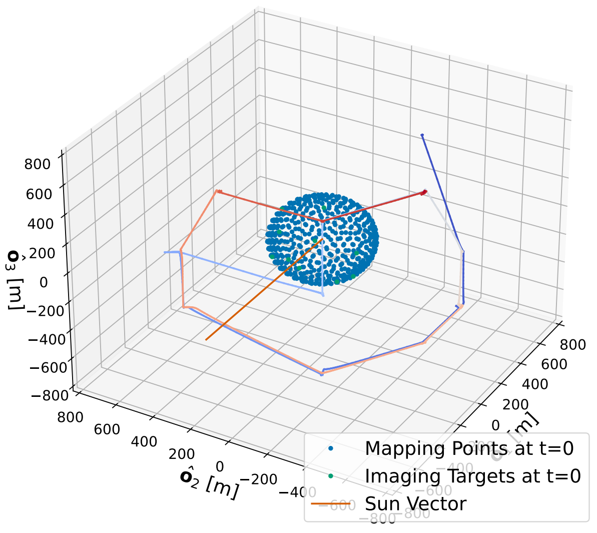
(b) Stored power with charging modes included.



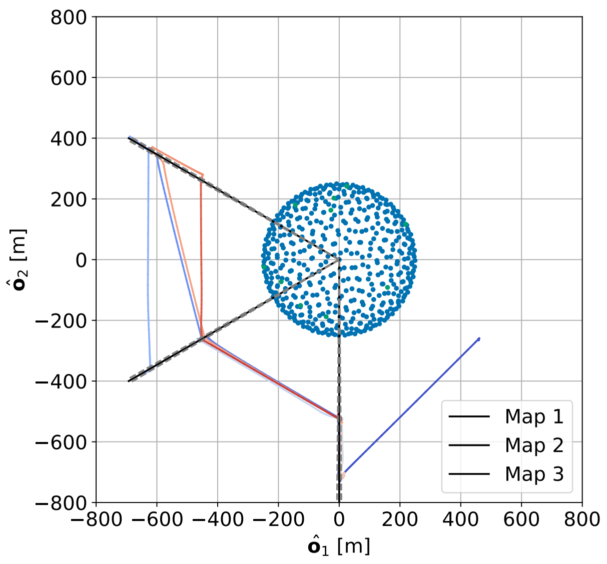
(c) Total ΔV consumed with maneuver modes included.

Figure 8.12: Spacecraft resources over time using optional navigation mode.

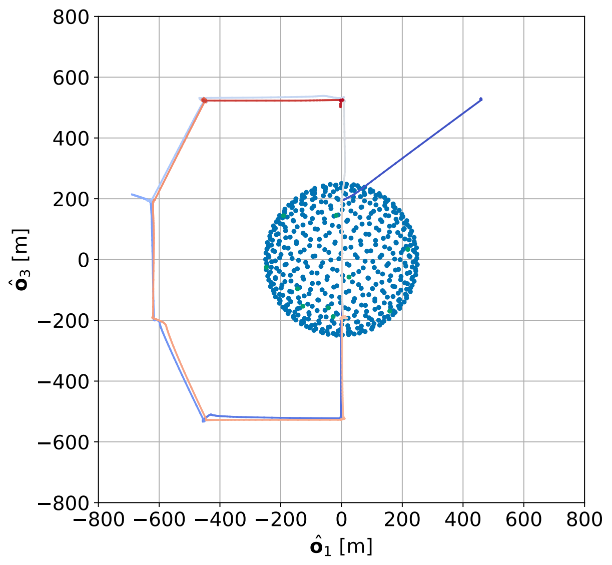




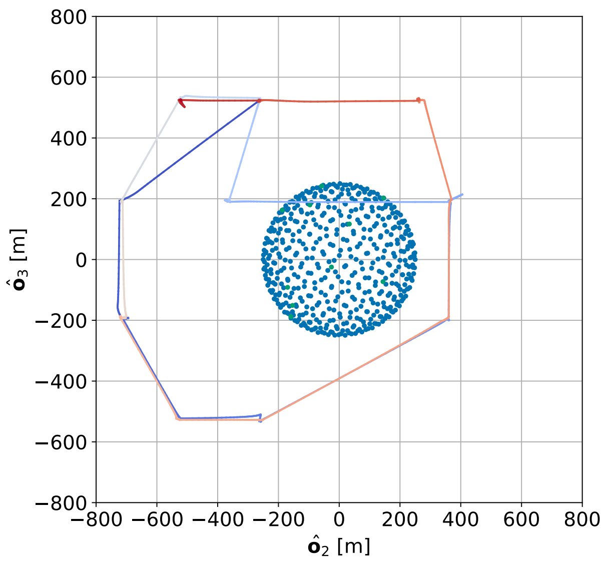
(a) 3D view.



(b) $\hat{\mathbf{o}}_1$ & $\hat{\mathbf{o}}_2$ plane.

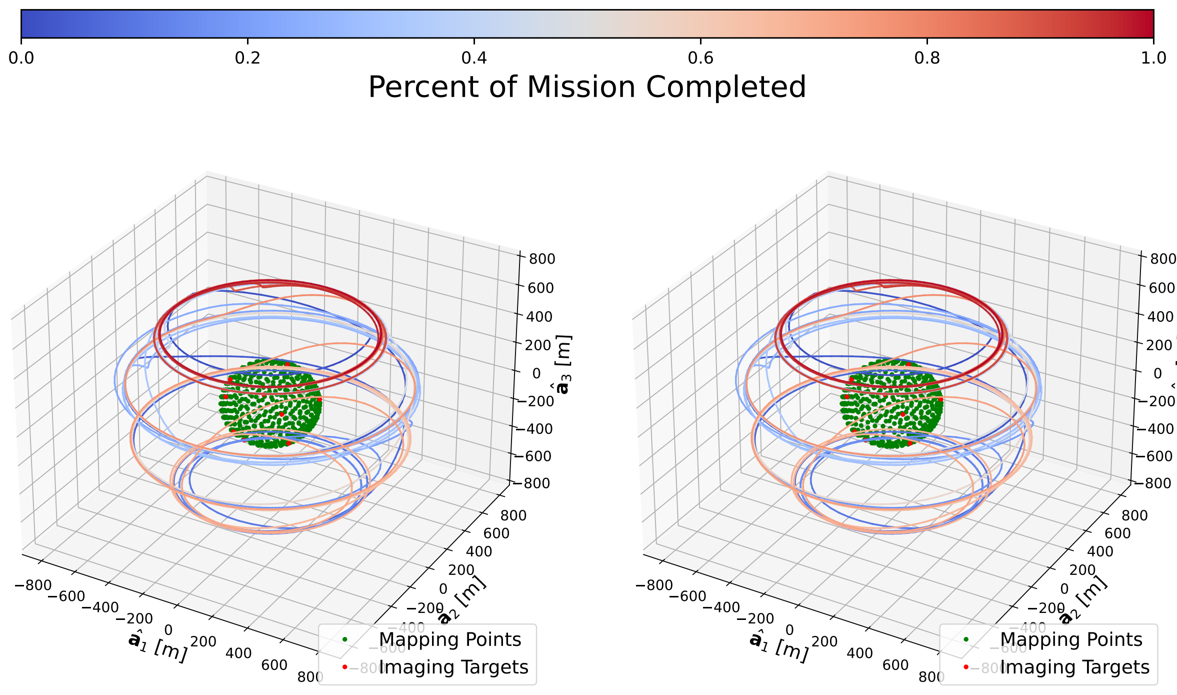


(c) $\hat{\mathbf{o}}_1$ & $\hat{\mathbf{o}}_3$ plane.



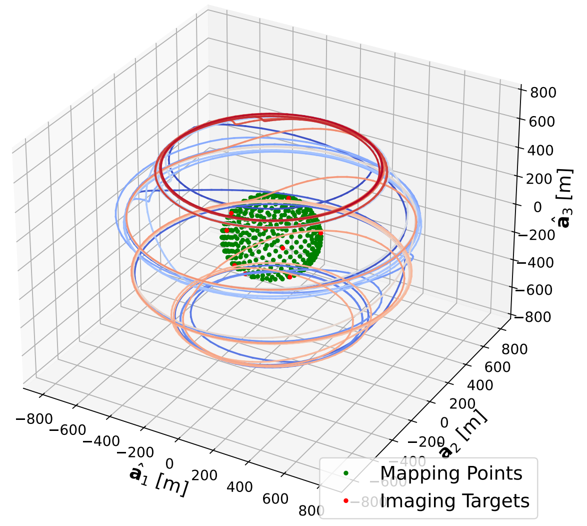
(d) $\hat{\mathbf{o}}_2$ & $\hat{\mathbf{o}}_3$ plane.

Figure 8.13: Hill frame trajectory with the navigation mode included in the action space.



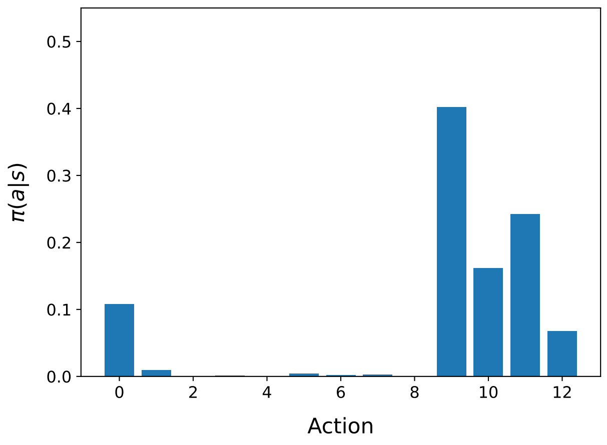
(a) 6 PM LST map.

(b) 2 PM LST map.

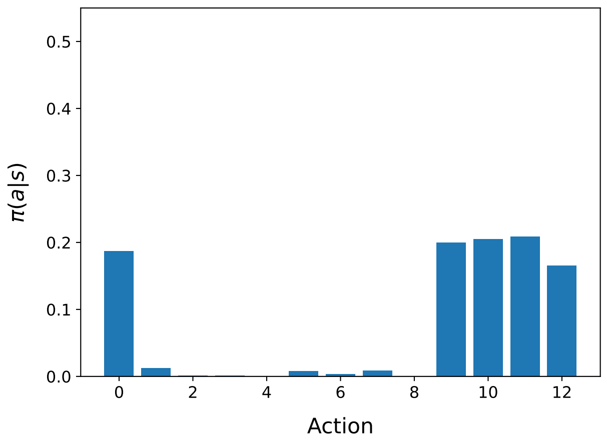


(c) 10 AM LST map.

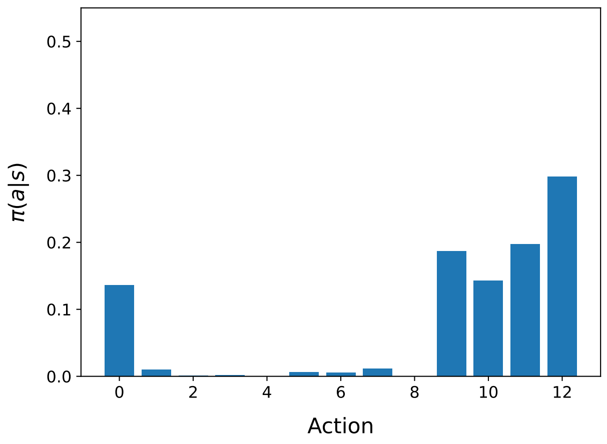
Figure 8.14: Trajectory of the spacecraft in the asteroid body frame. Green points represent mapping points collected by the spacecraft.



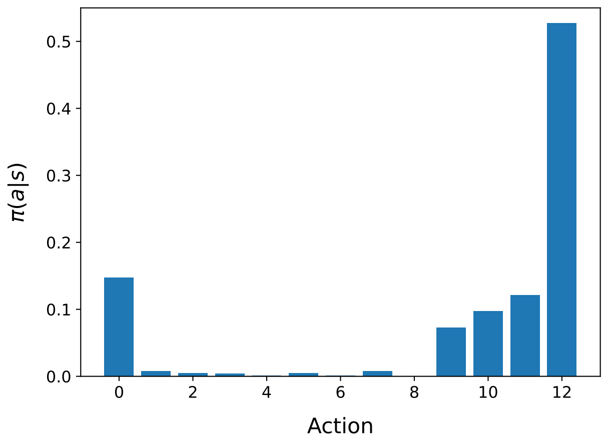
(a) $10^1 \leq \|\mathrm{diag}(P)\| < 10^2$



(b) $10^2 \leq \|\mathrm{diag}(P)\| < 10^3$



(c) $10^3 \leq \|\mathrm{diag}(P)\| < 10^4$



(d) $10^4 \leq \|\mathrm{diag}(P)\| < 10^5$

Figure 8.15: Dependency of stochastic policy outputs on the magnitude of the diagonal of the state covariance matrix. Action 0: Charge. Actions 1-8: Waypoint Maneuver. Action 9: Map. Action 10: Downlink. Action 11: Image Surface Target. Action 12: Navigation Update.

with no human intervention outside of notification that the window is no longer available. Finally, the decision-making agent is shown to be capable of managing its state estimate when an additional navigation mode is added to the action space. The decision-making agent is capable of producing policies that are equivalent to those trained without the navigation mode.

The aforementioned experiments demonstrate that reinforcement learning is a viable method for planning and scheduling of small body science operations, capable of managing spacecraft resources, maneuvers, and navigation updates while achieving the science objectives of the formulated mission. These results are promising, and future work should investigate using state-of-the-art autonomous guidance and relative navigation methods within this planning and scheduling paradigm for a real mission study. Furthermore, problem formulations that include multiple phases of operations should also be investigated.

Chapter 9

Conclusions

The majority of problem formulations for spacecraft planning and scheduling rely on simple models of the problem formulated as linear programs, which are insufficient to capture the complexity of a spacecraft operating in orbit about the Earth or within the proximity of an asteroid. The equations of motion governing the rotational and translational dynamics of a spacecraft are nonlinear. The insistence that planning and scheduling problems be formulated using linear programs for the sake of optimality guarantees ignores the fact that said optimality guarantees dissolve during actual operations due to nonlinearity, unmodeled or mismodeled dynamics, and uncertainties in the environment. This requires the use of replanning capabilities, which are relegated to repairing something that has already broken or responding to something that was not expected. As such, this thesis focuses on the application of reinforcement learning (RL) to planning and scheduling, which places no constraints on the problem formulation outside of the Markov assumption and opens the door to learning-based agents that can evolve their strategies over time. This chapter serves to highlight these contributions, and discuss what future work is required to make the use of reinforcement learning for spacecraft planning and scheduling ubiquitous. If humanity is to become a space faring civilization, we must untether ourselves from convenient mathematical representations and embrace the ability to learn from experience (simulated or real).

9.1 Major Contributions

This thesis addresses several large gaps in the literature, specifically regarding the use of reinforcement learning for Earth-observing satellite (EOS) scheduling. On-board data storage, downlink, and agile targeting of surface targets are all explored within the context of reinforcement learning and high-fidelity scheduling simulations, which has not yet been done in the literature. This work is presented in Chapters 3 and 4. Furthermore, a comprehensive comparison between various state-of-the-art reinforcement learning algorithms for EOS scheduling was performed, laying important groundwork for future work. These comparisons are presented in Chapter 5. The simulators and reinforcement learning algorithms developed (or interfaced with) in this thesis are open-source and available to the public in the bsk_rl repository.[1] I hope that future researchers will make use of this repository and add to these benchmark solutions. This work has also resulted in a minimum of 12 new Basilisk modules that are available to the public through the Basilisk repository,[2] many of which focus on on-board data handling, imaging, and mapping. These modules have helped Basilisk differentiate itself from other astrodynamics simulation packages, adding mission planning capabilities to its arsenal.

In addition to single satellite Earth-observing scheduling, this thesis presents a novel, robust, and scalable solution to the multi-satellite Earth-observing scheduling problem by applying policies trained with single agent reinforcement learning to the problem under various communication assumptions and target distribution methods. This work is presented in Chapter 6. A fully decentralized architecture is shown to be more performant than one that relies on an integer program for target distribution for the majority of cases. This supports the assertion that a reinforcement learning-based approach can outperform an integer program, especially if the problem is not fully modeled with the integer program. Furthermore, even though this method does not optimize a global reward function, it produces better solutions than a genetic algorithm that does optimize over a global reward function. While the genetic algorithm could be parameterized to produce

[1] https://github.com/AVSLab/bsk_rl
[2] https://github.com/AVSLab/basilisk

equivalent or better solutions, this would require days of computation for a single three-orbit planning horizon.

Finally, this thesis presents a novel application of reinforcement learning to the problem of small body science operations, demonstrating that a reinforcement learning policy is capable of autonomously managing maneuvers, navigation updates, resource management, and science operations to accomplish a mission. This work is presented in Chapters 7 and 8. The policies are shown to be robust to unanticipated events such as ground station outages. While various formulations of small body science operations problems have been investigated in the literature, none of these formulations handle all important facets of the problem from a planning and scheduling perspective, often focusing on guidance and control problems.

9.2 Future Work

While this thesis makes several important contributions to the field of spacecraft planning and scheduling, there are still many open questions that must be addressed before reinforcement learning can be ubiquitously adopted for spacecraft planning and scheduling. This section will highlight some of these open questions and discuss how they might be addressed in future work.

9.2.1 EOS Scheduling

Several open questions remain regarding the application of reinforcement learning to EOS scheduling. Long-term deployment of policies should be addressed further, especially in regard to safety and long-term degradation of spacecraft performance. A thorough comparison between integer programming and reinforcement learning is also worth pursuing. However, this study would likely take 6-12 months to complete, and deployment of both methods on a real physical system would be required to fully demonstrate the benefits of RL and ensure a straw man example is not constructed. This would only increase the length of the study, potentially requiring an entire PhD worth of work. I'm not sure that this is the best use of the entirety of someone's PhD. Lastly, and possibly most importantly, future work should apply multi-agent RL algorithms to a decentralized POMDP formulation of the multi-satellite agile Earth-observing scheduling problem. This would likely produce better performance than the method proposed in this work, but at the cost of scalability.

9.2.2 Small Body Science Operations

This thesis really only scratches the surface of the application of RL to small body science operations, largely because each component of the guidance, navigation, and control (GNC) subsystem is a PhD in and of itself. Significant collaboration between different students, labs, and/or government institutions is required to put forth a study that examines each phase of a representative small body mission from a planning and scheduling perspective with more representative GNC subsystems and science objectives. Furthermore, the question of exactly what RL enables from a mission architecting perspective remains open. Finally, while this thesis explores proximity operations about a small body, the application of this planning framework to rendezvous, proximity operations, and docking (RPOD) about Earth is evident. Like most of the literature exploring RL for small body proximity operations, the RPOD literature focuses on guidance and control problems. Equal emphasis should be placed on planning and scheduling problems.

9.2.3 Sim-To-Real Gap

I have suggested a comparison between RL and integer programming, deploying both methods on a real flight system to measure performance. The RL agents would be trained on high-fidelity simulations before deployment, and it is likely that some gap would exist between the RL simulator and the real spacecraft/environment. The impact of this gap, as well as techniques to mitigate its impact, should be explored further if this comparison work is pursued. Some work has demonstrated the successful closure of this gap for unmanned aerial systems, and I don't doubt that it is possible to close this gap for spacecraft either. However, it will be valuable to determine what may need to be included in the RL simulator to not require any transfer learning techniques during deployment.

Bibliography

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," Nature, vol. 550, pp. 354–359, 10 2017.

[2] "Satellite database." Union of Concerned Scientists: https://www.ucsusa.org/resources/satellite-database, 2022.

[3] T. J. Muelhaupt, M. E. Sorge, J. Morin, and R. S. Wilson, "Space traffic management in the new space era," Journal of Space Safety Engineering, vol. 6, no. 2, pp. 80–87, 2019.

[4] J. C. McDowell, "The low earth orbit satellite population and impacts of the spacex starlink constellation," The Astrophysical Journal Letters, vol. 892, no. 2, p. L36, 2020.

[5] N. Pachler, I. del Portillo, E. F. Crawley, and B. G. Cameron, "An updated comparison of four low earth orbit satellite constellation systems to provide global broadband," in 2021 IEEE international conference on communications workshops (ICC workshops), pp. 1–7, IEEE, 2021.

[6] "Audit of nasa's deep space network," tech. rep., NASA Office of Inspector General, 2023.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[8] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al., "Dota 2 with large scale deep reinforcement learning," arXiv preprint arXiv:1912.06680, 2019.

[9] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood, "Casper: Space exploration through continuous planning," IEEE Intelligent Systems, vol. 16, pp. 70–75, September 2001.

[10] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan, "Towards an application framework for automated planning and scheduling," in 1997 IEEE Aerospace Conference, pp. 375–386, April 1997.

[11] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davis, D. Mandl, S. Frye, B. Trout, S. Shulman, and D. Boyer, "Using autonomy flight software to improve science return on earth observing one," Journal of Aerospace Computing, Information, and Communication, vol. 2, no. 4, pp. 196–216, 2005.

[12] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye, "Timeline-based space operations scheduling with external constraints," in Twentieth International Conference on Automated Planning and Scheduling, 2010.

[13] S. Chien, J. Doubleday, D. R. Thompson, K. Wagstaff, J. Bellardo, C. Francis, E. Baumgarten, A. Williams, E. Yee, E. Stanton, and J. Piug-Suari, "Onboard autonomy on the intelligent payload experiment (ipex) cubesat mission," Journal of Aerospace Information Systems (JAIS), April 2016.

[14] S. A. Chien, A. G. Davies, J. Doubleday, D. Q. Tran, D. Mclaren, W. Chi, and A. Maillard, "Automated volcano monitoring using multiple space and ground sensors," Journal of Aerospace Information Systems, vol. 17, no. 4, pp. 214–228, 2020.

[15] S. Chien, D. Mclaren, J. Doubleday, D. Tran, V. Tanpipat, and R. Chitradon, "Using taskable remote sensing in a sensor web for thailand flood monitoring," Journal of Aerospace Information Systems, vol. 16, no. 3, pp. 107–119, 2019.

[16] V. Verma, D. Gaines, G. Rabideau, S. Schaffer, and R. Joshi, "Autonomous science restart for the planned europa mission with lightweight planning and execution," in International Workshop on Planning and Scheduling for Space (IWPSS 2017), (Pittsburgh, PA), June 2017.

[17] D. Gaines, S. Chien, G. Rabideau, S. Kuhn, V. Wong, A. Yelamanchili, S. Towey, J. Agrawal, W. Chi, A. Connell, E. Davis, and C. Lohr, "Onboard planning for the mars 2020 perseverance rover," in 16th Symposium on Advanced Space Technologies in Robotics and Automation, June 2022.

[18] A. K. Kennedy, Planning and Scheduling for Earth-Observing Small Satellite Constellations. PhD thesis, Massachusetts Institute of Technology, 2018.

[19] M. Dahl, J. Chew, and K. Cahoy, "Optimization of smallsat constellations and low cost hardware to utilize onboard planning," in ASCEND 2021, p. 4172, 2021.

[20] X. Wang, G. Wu, L. Xing, and W. Pedrycz, "Agile earth observation satellite scheduling over 20 years: Formulations, methods, and future directions," IEEE Systems Journal, vol. 15, no. 3, pp. 3881–3892, 2020.

[21] A. Globus, J. Crawford, J. Lohn, R. Morris, and D. Clancy, "Scheduling earth observing fleets using evolutionary algorithms: Problem description and approach," International Conference on Space Mission Challenges for Information Technology, 2003.

[22] S. Spangelo, J. Cutler, K. Gilson, and A. Cohn, "Optimization-based scheduling for the single-satellite, multi-ground station communication problem," Computers and Operations Research, vol. 57, May 2015.

[23] D.-H. Cho, J.-H. Kim, H.-L. Choi, and J. Ahn, "Optimization-based scheduling method for agile earth-observing satellite constellation," Journal of Aerospace Information Systems, vol. 15, no. 11, pp. 611–626, 2018.

[24] J. Kim, J. Ahn, H.-L. Choi, and D.-H. Cho, "Task scheduling of agile satellites with transition time and stereoscopic imaging constraints," Journal of Aerospace Information Systems, vol. 17, no. 6, pp. 285–293, 2020.

[25] X. Chen, G. Reinelt, G. Dai, and A. Spitz, "A mixed integer linear programming model for multi-satellite scheduling," European Journal of Operational Research, vol. 275, no. 2, pp. 694–707, 2019.

[26] P. Monmousseau, "Scheduling of a constellation of satellites: Creating a mixed-integer linear model," Journal of Optimization Theory and Applications, vol. 191, no. 2-3, pp. 846–873, 2021.

[27] J. Kim and J. Ahn, "Integrated framework for task scheduling and attitude control of multiple agile satellites," Journal of Aerospace Information Systems, vol. 18, no. 8, pp. 539–552, 2021.

[28] J. Cappaert, F. Foston, P. S. Heras, B. King, N. Pascucci, J. Reilly, C. Brown, J. Pitzo, and M. Tallhamm, "Constellation modelling, performance prediction and operations management for the spire constellation," in SmallSat Conference, 2021.

[29] V. Shah, V. Vittaldev, L. Stepan, and C. Foster, "Scheduling the world's largest earth-observing fleet of medium-resolution imaging satellites," in International Workshop on Planning and Scheduling for Space, 2019.

[30] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, "Mixed-integer nonlinear optimization," Acta Numerica, vol. 22, pp. 1–131, 2013.

[31] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza, "Reaching the limit in autonomous racing: Optimal control versus reinforcement learning," Science Robotics, vol. 8, no. 82, p. eadg1462, 2023.

[32] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[33] P. Blacker, C. P. Bridges, and S. Hadfield, "Rapid prototyping of deep learning models on radiation hardened cpus," in 2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 25–32, IEEE, 2019.

[34] E. Dunkel, J. Swope, Z. Towfic, S. Chien, D. Russell, J. Sauvageau, D. Sheldon, J. Romero-Cañas, J. L. Espinosa-Aranda, L. Buckley, et al., "Benchmarking deep learning inference of remote sensing imagery on the qualcomm snapdragon and intel movidius myriad x processors onboard the international space station," in IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium, pp. 5301–5304, IEEE, 2022.

[35] R. Furfaro, A. Scorsoglio, R. Linares, and M. Massari, "Adaptive generalized zem-zev feedback guidance for planetary landing via a deep reinforcement learning approach," Acta Astronautica, vol. 171, pp. 156–171, 2020.

[36] B. Gaudet, R. Linares, and R. Furfaro, "Adaptive guidance and integrated navigation with reinforcement meta-learning," Acta Astronautica, vol. 169, pp. 180–190, April 2020.

[37] B. Gaudet, R. Linares, and R. Furfaro, "Deep reinforcement learning for six degree-of-freedom planetary landing," Advances in Space Research, vol. 65, no. 7, pp. 1723–1741, 2020.

[38] A. Scorsoglio, A. D'Ambrosio, L. Ghilardi, B. Gaudet, F. Curti, and R. Furfaro, "Image-based deep reinforcement meta-learning for autonomous lunar landing," Journal of Spacecraft and Rockets, vol. 59, no. 1, pp. 153–165, 2022.

[39] B. Gaudet, R. Linares, and R. Furfaro, "Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations," Acta Astronautica, vol. 171, pp. 1–13, 2020.

[40] B. Gaudet, R. Linares, and R. Furfaro, "Six degree-of-freedom body-fixed hovering over unmapped asteroids via lidar altimetry and reinforcement meta-learning," Acta Astronautica, vol. 172, pp. 90–99, 2020.

[41] S. Takahashi and D. Scheeres, "Autonomous proximity operations at small neas," in 33rd International Symposium on Space Technology and Science (ISTS), vol. 2, 2022.

[42] L. Federici, A. Scorsoglio, L. Ghilardi, A. D'Ambrosio, B. Benedikter, A. Zavoli, and R. Furfaro, "Image-based meta-reinforcement learning for autonomous guidance of an asteroid impactor," Journal of Guidance, Control, and Dynamics, vol. 45, no. 11, pp. 2013–2028, 2022.

[43] K. Hovell and S. Ulrich, "Deep reinforcement learning for spacecraft proximity operations guidance," Journal of Spacecraft and Rockets, vol. 58, no. 2, pp. 254–264, 2021.

[44] K. Hovell and S. Ulrich, "Laboratory experimentation of spacecraft robotic capture using deep-reinforcement-learning–based guidance," Journal of Guidance, Control, and Dynamics, vol. 45, no. 11, pp. 2138–2146, 2022.

[45] L. Federici, B. Benedikter, and A. Zavoli, "Deep learning techniques for autonomous spacecraft guidance during proximity operations," Journal of Spacecraft and Rockets, vol. 58, no. 6, pp. 1774–1785, 2021.

[46] C. E. Oestreich, R. Linares, and R. Gondhalekar, "Autonomous six-degree-of-freedom spacecraft docking with rotating targets via reinforcement learning," Journal of Aerospace Information Systems, vol. 18, no. 7, pp. 417–428, 2021.

[47] L. Federici, A. Scorsoglio, A. Zavoli, and R. Furfaro, "Meta-reinforcement learning for adaptive spacecraft guidance during finite-thrust rendezvous missions," Acta Astronautica, vol. 201, pp. 129–141, 2022.

[48] K. Dunlap, M. Mote, K. Delsing, and K. L. Hobbs, "Run time assured reinforcement learning for safe satellite docking," Journal of Aerospace Information Systems, vol. 20, no. 1, pp. 25–36, 2023.

[49] W. Haijiao, Y. Zhen, Z. Wugen, and L. Dalin, "Online scheduling of image satellites based on neural networks and deep reinforcement learning," Chinese Journal of Aeronautics, vol. 32, no. 4, pp. 1011–1019, 2019.

[50] X. Zhao, Z. Wang, and G. Zheng, "Two-phase neural combinatorial optimization with reinforcement learning for agile satellite scheduling," Journal of Aerospace Information Systems, vol. 17, no. 7, pp. 346–357, 2020.

[51] Y. He, L. Xing, Y. Chen, W. Pedrycz, L. Wang, and G. Wu, "A generic markov decision process model and reinforcement learning method for scheduling agile earth observation satellites," IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020.

[52] A. Harris, T. Valade, T. Teil, and H. Schaub, "Generation of spacecraft operations procedures using deep reinforcement learning," Journal of Spacecraft and Rockets, pp. 1–16, 2021.

[53] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

[54] D. Eddy and M. Kochenderfer, "Markov decision processes for multi-objective satellite task planning," in 2020 IEEE Aerospace Conference, pp. 1–12, IEEE, 2020.

[55] M. J. Kochenderfer, Decision Making Under Uncertainty: Theory and Application, ch. Sequential Problems, pp. 102–103. Massachusetts Institute of Technology, 2015.

[56] M. J. Kochenderfer, T. A. Wheeler, and K. H. Wray, Algorithms for Decision Making. MIT Press, 2022.

[57] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved monte-carlo search," Univ. Tartu, Estonia, Tech. Rep, 2006.

[58] D. Shah, Q. Xie, and Z. Xu, "Non-asymptotic analysis of monte carlo tree search," in Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '20, (New York, NY, USA), pp. 31–32, Association for Computing Machinery, 2020.

[59] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in International conference on machine learning, pp. 1928–1937, PMLR, 2016.

[60] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[61] A. T. Harris, Autonomous Management and Control of Multi-Spacecraft Operations Leveraging Atmospheric Forces. PhD thesis, University of Colorado at Boulder, 2021.

[62] A. Hadj-Salah, J. Guerra, M. Picard, and M. Capelle, "Towards operational application of deep reinforcement learning to earth observation satellite scheduling," 2020.

[63] A. Herrmann and H. Schaub, "Autonomous on-board planning for earth-orbiting spacecraft," in IEEE Aerospace Conference, (Big Sky, MT), March 5-12 2022.

[64] A. Herrmann and H. Schaub, "Reinforcement learning for the agile earth-observing satellite scheduling problem," IEEE Transactions on Aerospace and Electronic Systems, pp. 1–13, 2023.

[65] P. W. Kenneally, S. Piggott, and H. Schaub, "Basilisk: A flexible, scalable and modular astrodynamics simulation framework," Journal of Aerospace Information Systems, vol. 17, pp. 496–507, Sept. 2020.

[66] G. S. Center, "Near earth network users' guide," tech. rep., National Aeronautics and Space Administration, Greenbelt, MD, March 2019.

[67] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., "Mastering the game of go with deep neural networks and tree search," nature, vol. 529, no. 7587, pp. 484–489, 2016.

[68] A. Herrmann and H. Schaub, "Autonomous spacecraft tasking using monte carlo tree search methods," in Proceedings of the AAS/AIAA Space Flight Mechanics Meeting, Charlotte, NC, USA, pp. 1–4, 2021.

[69] A. P. Herrmann and H. Schaub, "Monte carlo tree search methods for the earth-observing satellite scheduling problem," Journal of Aerospace Information Systems, pp. 1–13, 2021.

[70] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, no. 56, pp. 1929–1958, 2014.

[71] M. Gustineli, "A survey on recently proposed activation functions for deep learning," arXiv preprint arXiv:2204.02921, 2022.

[72] A. Herrmann and H. Schaub, "A comparison of deep reinforcement learning algorithms for earth-observing satellite scheduling," in AAS Spaceflight Mechanics Meeting, (Austin, TX), Jan. 15–19 2023. Paper No. AAS 23-116.

[73] H. Schaub and J. L. Junkins, Analytical Mechanics of Space Systems. Reston, VA: AIAA Education Series, 4th ed., 2018.

[74] K. Cui, J. Song, L. Zhang, Y. Tao, W. Liu, and D. Shi, "Event-triggered deep reinforcement learning for dynamic task scheduling in multi-satellite resource allocation," IEEE Transactions on Aerospace and Electronic Systems, 2022.

[75] L. Dalin, W. Haijiao, Y. Zhen, G. Yanfeng, and S. Shi, "An online distributed satellite cooperative observation scheduling algorithm based on multiagent deep reinforcement learning," IEEE Geoscience and Remote Sensing Letters, vol. 18, no. 11, pp. 1901–1905, 2020.

[76] Y. Wang, M. Damani, P. Wang, Y. Cao, and G. Sartoretti, "Distributed reinforcement learning for robot teams: A review," Current Robotics Reports, vol. 3, no. 4, pp. 239–257, 2022.

[77] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in Proceedings of the Tenth International Conference on Machine Learning, pp. 330–337, 1993.

[78] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Comparative evaluation of cooperative multi-agent deep reinforcement learning algorithms," arXiv preprint arXiv:2006.07869, 2020.

[79] C. S. de Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. Torr, M. Sun, and S. Whiteson, "Is independent learning all you need in the starcraft multi-agent challenge?," arXiv preprint arXiv:2011.09533, 2020.

[80] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," Advances in neural information processing systems, vol. 30, 2017.

[81] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in Proceedings of the AAAI conference on artificial intelligence, vol. 32, 2018.

[82] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al., "Value-decomposition networks for cooperative multi-agent learning," arXiv preprint arXiv:1706.05296, 2017.

[83] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," The Journal of Machine Learning Research, vol. 21, no. 1, pp. 7234–7284, 2020.

[84] F. A. Oliehoek and C. Amato, A Concise Introduction to Decentralized POMDPs. Springer, 2016.

[85] C. Boutilier, "Sequential optimality and coordination in multiagent systems," in IJCAI, vol. 99, pp. 478–485, 1999.

[86] A. Herrmann and H. Schaub, "Reinforcement learning for the multi-satellite earth-observing scheduling problem," in AAS Guidance and Controls Conference, (Breckenridge, CO), Feb. 3-9 2022.

[87] A. Herrmann, M. Stephenson, and H. Schaub, "Reinforcement learning for multi-satellite agile earth observing scheduling under various communication assumptions," in AAS Guidance and Control Conference, (Breckenridge, CO), Feb. 2–8 2023. Paper No. AAS-23-146.

[88] S. A. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye, "Improving the operations of the earth observing one mission via automated mission planning," 2010.

[89] A. Yelamanchili, G. Rabideau, J. Agrawal, V. Wong, D. Gaines, S. Chien, E. Fosse, J. Biehl, S. Kuhn, A. Connell, et al., "Ground and onboard automated scheduling for the mars 2020 rover mission," International Workshop on Planning and Scheduling for Space, 2021.

[90] G. Rabideau, V. Wong, D. Gaines, J. Agrawal, S. Chien, E. Fosse, and J. Biehl, "Onboard automated scheduling for the mars 2020 rover," 2020.

[91] D. M. Chan and A. Agha-mohammadi, "Autonomous imaging and mapping of small bodies using deep reinforcement learning," in 2019 IEEE Aerospace Conference, pp. 1–12, 2019.

[92] M. Piccinin, P. Lunghi, and M. Lavagna, "Deep reinforcement learning-based policy for autonomous imaging planning of small celestial bodies mapping," Aerospace Science and Technology, vol. 120, p. 107224, 2022.

[93] S. Takahashi and D. Scheeres, "Autonomous proximity operations at small neas," 33rd International Symposium on Space Technology and Science (ISTS), 02 2022.

[94] A. Herrmann and H. Schaub, "Reinforcement learning for small body science operations," in AAS/AIAA Astrodynamics Specialist Conference, (Charlotte, NC), Aug. 7-11 2022.

[95] V. Pesce, A.-a. Agha-mohammadi, and M. Lavagna, "Autonomous navigation & mapping of small bodies," in IEEE Aerospace Conference, pp. 1–10, IEEE, 2018.

[96] I. A. Nesnas, B. J. Hockman, S. Bandopadhyay, B. J. Morrell, D. P. Lubey, J. Villa, D. S. Bayard, A. Osmundson, B. Jarvis, M. Bersani, et al., "Autonomous exploration of small bodies toward greater autonomy for deep space missions," Frontiers in Robotics and AI, vol. 8, 2021.

[97] M. Ashman, M. Barthélémy, M. Almeida, N. Altobelli, M. C. Sitjà, J. J. G. Beteta, B. Geiger, B. Grieger, D. Heather, R. Hoofs, et al., "Rosetta science operations in support of the philae mission," Acta Astronautica, vol. 125, pp. 41–64, 2016.

[98] D. Lauretta, S. Balram-Knutson, E. Beshore, W. Boynton, C. D. d'Aubigny, D. DellaGiustina, H. Enos, D. Golish, C. Hergenrother, E. Howell, et al., "Osiris-rex: Sample return from asteroid (101955) bennu," Space Science Reviews, vol. 212, no. 1, pp. 925–984, 2017.

[99] D. J. Scheeres, "Orbit mechanics about asteroids and comets," Journal of Guidance, Control, and Dynamics, vol. 35, no. 3, pp. 987–997, 2012.

[100] S. Takahashi and D. J. Scheeres, "Autonomous exploration of a small near-earth asteroid," Journal of Guidance, Control, and Dynamics, vol. 44, no. 4, pp. 701–718, 2021.

[101] D. Simon, Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. John Wiley & Sons, 2006.

[102] S. Bhaskaran, S. Desai, P. Dumont, B. Kennedy, G. Null, W. Owen Jr, J. Riedel, S. Synnott, and R. Werner, "Orbit determination performance evaluation of the deep space 1 autonomous navigation system," 1998.

[103] R. Gaskell, O. Barnouin-Jha, D. J. Scheeres, A. Konopliv, T. Mukai, S. Abe, J. Saito, M. Ishiguro, T. Kubota, T. Hashimoto, et al., "Characterizing and navigating small bodies with imaging data," Meteoritics & Planetary Science, vol. 43, no. 6, pp. 1049–1061, 2008.

[104] A. Konopliv, S. Asmar, R. Park, B. Bills, F. Centinello, A. Chamberlin, A. Ermakov, R. Gaskell, N. Rambaux, C. Raymond, et al., "The vesta gravity field, spin pole and rotation period, landmark positions, and ephemeris from the dawn tracking and optical data," Icarus, vol. 240, pp. 103–117, 2014.

[105] B. Williams, P. Antreasian, E. Carranza, C. Jackman, J. Leonard, D. Nelson, B. Page, D. Stanbridge, D. Wibben, K. Williams, et al., "Osiris-rex flight dynamics and navigation design," Space Science Reviews, vol. 214, pp. 1–43, 2018.

[106] A. Dietrich and J. W. McMahon, "Orbit determination using flash lidar around small bodies," Journal of Guidance, Control, and Dynamics, vol. 40, no. 3, pp. 650–665, 2017.

[107] J. C. Sanchez and H. Schaub, "Small body navigation and gravity estimation using kalman filter and least-squares fitting," in AAS Spaceflight Mechanics Meeting, (Austin, TX), Jan. 15–19 2023. Paper No. AAS 23-126.

[108] B. Schutz, B. Tapley, and G. H. Born, Statistical orbit determination. Elsevier, 2004.

[109] P. W. Kenneally et al., "Basilisk: A flexible, scalable and modular astrodynamics simulation framework," in 7th International Conference on Astrodynamics Tools and Techniques (ICATT), (DLR Oberpfaffenhofen, Germany), Nov. 6–9 2018.

[110] D. Lauretta, A. Bartels, M. Barucci, E. Bierhaus, R. Binzel, W. Bottke, H. Campins, S. Chesley, B. Clark, B. Clark, et al., "The osiris-rex target asteroid (101955) bennu: Constraints on its physical, geological, and dynamical nature from astronomical observations," Meteoritics & Planetary Science, vol. 50, no. 4, pp. 834–849, 2015.

[111] D. Scheeres, J. McMahon, A. French, D. Brack, S. Chesley, D. Farnocchia, Y. Takahashi, J. Leonard, J. Geeraert, B. Page, et al., "The dynamic geophysical environment of (101955) bennu based on osiris-rex measurements," Nature Astronomy, vol. 3, no. 4, pp. 352–361, 2019.

[112] L. Shekhtman, "Osiris-rex engineers pull off a daring rescue of a monumental asteroid mission," 2019. [Online; posted 10-September-2019].