Advancing Satellite Autonomy:
Methods for Earth Observation & Spacecraft Inspection

by

Mark Andrew Stephenson

B.S. Mechanical Engineering, University of Southern California, 2022

M.S. Aerospace Engineering Sciences, University of Colorado Boulder, 2024

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

2026

Committee Members:

Prof. Hanspeter Schaub

Dr. Jeremy Frank

Prof. Morteza Lahijanian

Prof. John Martin

Prof. Zachary Sunberg

Stephenson, Mark Andrew (Ph.D., Aerospace Engineering Sciences)

Advancing Satellite Autonomy: Methods for Earth Observation & Spacecraft Inspection

Thesis directed by Prof. Hanspeter Schaub

As the quantity and capabilities of satellites have increased, the primary challenges in mission operations have evolved from piloting a single spacecraft to simultaneously managing multiple complex objectives across a fleet of satellites. Historically, satellite operations have relied on a cycle of ground-based planning and on-orbit plan execution. However, many subsystems have matured to a level of reliability that requires minimal manual supervision, and the satellite-to-operator ratio is inverting, making per-satellite operation impractical. Simultaneously, advances in onboard data collection and processing enable production of actionable information *in situ*. These factors enable—and even necessitate—a paradigm shift in satellite operations from ground-generated plans toward closed-loop autonomy with responsive policies. With a policy-based framework, individual spacecraft can autonomously optimize across individual and constellation-wide mission objectives and proactively respond to unexpected events, advantageous or otherwise, without ground intervention. While reinforcement learning (RL) has enabled similar capabilities in other domains, its application to aerospace remains limited due to stricter standards for technology adoption.

This work advances the application of RL for satellite autonomy in two classes of problems. For Earth observation, an RL-based approach to the agile Earth-observing satellite scheduling problem (AEOSSP) is developed and compared to mixed-integer optimization methods in realistic settings. The capabilities are extended to yield closed-loop responsiveness for constellation-wide scheduling and distributed tip-and-cue operations. For resident space object (RSO) inspection, a formulation with impulsive maneuvering is introduced and solved using RL and path planning algorithms. Both produce low-Δv solutions by utilize natural motion trajectories; analytical safety guarantees are also enforced. Across these applications, RL yields closed-loop autonomy that performs competitively with other methods in realistic and relevant space missions.

Dedication

Remembering Granddad, Dr. Jack Stephenson.

Yet aloft, in Map-space, origins, destinations, any Termini, hardly seem to matter,— one can apprehend all at once the entire plexity of possible journeys, set as one is above Distance, above Time itself. Thomas Pynchon, *Mason & Dixon*, p. 669

Acknowledgements

First, thank you to my advisor Dr. Hanspeter Schaub, who has given me both freedom to explore and steady guidance to navigate me towards the terminus of the program. I appreciate that you took a chance on me by hiring someone with no research experience and little idea about what I wanted to study; being part of your lab quickly fixed both of those problems.

Over the past four years, the members of AVS lab have become central to my life as mentors, colleagues, and friends. The time we have spent together has truly been the best and most unforgettable part of this experience, from ski days to game nights and everything in between (including some work!). In particular (and because a comprehensive list wouldn't fit), thank you to John Martin for encouraging me to persevere through the challenges of the Ph.D. and Lorenzzo Mantovani for endlessly acting as a sounding board and collaborator.

I must also thank NASA and the NSTGRO program for supporting my work. Great nations invest in the sciences and their researchers; yet, annual NSTGRO awards have dropped by half since my grant began. From this program, I would like to acknowledge my mentors at Ames Research Center (Jeremy Frank) and JPL (Federico Rossi, J.P. de la Croix, and Amir Rahmani). To João, Juan, Robyn, Kristen, and Ben, our lunches on the 301 patio will not be forgotten.

Finally, I would not have gotten here without the unwavering support and encouragement of my family. Mom, Dad, and Nick: Each of you have been role models since the beginning. To David, thank you for being by my side from the second week of the Ph.D. through all that has followed.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

1.1 Trends in Satellite Operations

As the quantity and capabilities of satellites have increased, the primary challenges in mission operations have evolved from piloting a single spacecraft to simultaneously managing multiple complex objectives across a fleet of satellites. Historically, satellite operations have relied on a cycle of ground-based planning and on-orbit plan execution. However, many subsystems have matured to a level of reliability that requires minimal manual supervision, and the satellite-to-operator ratio is inverting, making per-satellite operation impractical [1]. Simultaneously, advances in onboard data collection and processing enable production of actionable information *in situ* [2, 3]. These factors enable—and even necessitate—a paradigm shift in satellite operations from ground-generated plans toward closed-loop autonomy. One form of autonomy are policies, or functions that map states to actions in order to produce some desirable behavior in an environment in a closed-loop fashion. With a policy-based framework, individual spacecraft can autonomously optimize across individual and constellation-wide mission objectives and proactively respond to unexpected events, advantageous or otherwise, without ground intervention. Reinforcement learning (RL) is one approach for generating policies, in which they are learned through exploratory interaction with a black-box environment. While RL has enabled similar capabilities in other domains, its application to aerospace remains limited due to stricter standards for technology adoption. This work advances the application of RL for satellite autonomy in two classes of problems: Earth observation and relative-motion-based resident space object (RSO) inspection.

(a) Planet Labs' Dove and SkySat Earth observation constellations [4].

(b) Starfish Space's Small Spacecraft Propulsion and Inspection Capability (SSPICY) inspection mission [5].

Figure 1.1: Examples of space-based observation missions.

Among the categories of missions being carried out today and the near future, observation tasks stand out for their potential to be improved by autonomy. This class includes planetary observation tasks, such as Earth-observing constellation scheduling (Figure 1.1a) for scientific, commercial, and military purposes, as well as on-orbit inspection tasks (Figure 1.1b) for servicing, debris analysis, and intelligence. These missions are characterized by complex ground-based planning and the potential for new information during execution that impacts the performance of the plan. The two factors are incompatible: Responsiveness cannot be achieved if there is always at least a ground communication cycle of delay when updating plans. In general, autonomous onboard decision-making is required to maximize the utility of any mission where information appears locally that impacts the optimality of upcoming decisions faster than an open-loop planning cycle is able to account for.

Distributed autonomy for Earth-observing constellations is becoming more viable and necessary due to three major trends:

(1) Research on ground-based preplanning approaches has slowed because of several insurmountable factors that must practically be considered for the agile Earth-observing satel-

lite scheduling problem (AEOSSP) [6, 7]. For perfect knowledge cases, these approaches formulate and solve the discrete optimization problem using mixed-integer linear programs (MILPs) or iterative local search (ILS), both facing computational complexity issues as the number of satellites and requests increases. These frameworks also make considering nonlinear resource constraints and other additional factors challenging or impossible. However, the greater issue is that ground-based preplanning methods inherently produce brittle schedules, so solution methods that only apply to the perfect knowledge case have limited practical applicability. Since updates to plans in the non-autonomous paradigm are subject to ground station accessibility, opportunistic or unexpected events cannot be effectively handled on-the-fly by any methods of this class [3]. Furthermore, these methods general are too computationally complex to execute on flight computers, meaning that onboard autonomy likely requires the application of different classes of methods. The limitations of these solvers are further discussed in the literature review in subsection 1.2.2.

(2) The cost of launch services and satellite hardware has decreased considerably. This trend is most apparent in cubesats and smallsats, which have become more capable and accessible [2]. As a result, observation satellites and constellations are proliferating; examples include Planet's Dove constellation [8], Maxar's WorldView satellites [9], and the Space Development Agency's planned tracking layer [10]. A larger number of increasingly capable satellites increases the potential for exploiting responsive behaviors; however, it can pose greater workloads for operators and ground-based planners if manual operations are necessary for all satellites.

(3) Scientists and other data consumers have identified distributed autonomy as a way of improving satellite and constellation data products. The current paradigm for many Earth observation missions is one or few satellites equipped with nadir scanning instruments; while this reduces the risk of missing the collection of data of interest due to poor tasking, it limits data collection to an orbital cadence and within the capabilities of a generally

wide-field of view (FOV) instrument. Reactivity introduces the potential for missteps, but enables significantly more efficient architectures by focusing operations on high priority tasks. An early example of using onboard autonomy to improve observation data is the Earth Observing-1 (EO-1)'s Autonomous Sciencecraft Experiment, in which onboard replanning was used to achieve high-level goals with onboard feature recognition [11]. A number of NASA groups produced frameworks for onboard reasoning and continuous reasoning during this period include Remote Agent [12], ASPEN [13], and Casper [14]. More recently, NASA's New Observing Strategy has extended this concept to a federation of spacecraft and ground-based data source sharing information to observe with real-time responsivity [15]. JPL's dynamic targeting method utilizes this approach at a small scale for cloud avoidance with an agile instrument [16, 17]. Across many satellites, this could extend to reactive targeting of emergent phenomena; this type of operation is being considered by NASA's Distributed Spacecraft Autonomy projects [18].

Together, these factors point to autonomous, distributed systems as the future of Earth observation both to improve current performance and enable new capabilities.

Spacecraft inspection is a nascent but rapidly growing category of missions as rendezvous and proximity operations (RPO) and in-space assembly and manufacturing (ISAM) concepts become more prevalent. Inspection is needed to characterize RSOs prior to docking, servicing, deorbiting, or other operations. One of the earliest examples of an inspection mission was completed by the XXS-10 microsatellite [19]. As the field develops, autonomous methods are emerging as a potential solution for the task [20, 21]. The growth of RPO is reflected in the diversity of planned and current missions: Northrop Grumman's Mission Extension Vehicle has been demonstrated, as the name implies, as a means of extending the lifetime of a known space asset [22]. Inspecting the target spacecraft for active deorbiting missions such as Astroscale's ELSA-M is more critical, as the defunct satellite may have unknown physical or dynamical properties [23]. The AFRL's Safe Trusted Autonomy for Responsible Spacecraft (STARS) program is concerned with inspecting

an unknown (and potentially adversarial) spacecraft [24], while Starfish Space's SSPICY mission offers the first instance of an in-space, RSO-observation-focused technology demonstration [5]. Planning trajectories for these inspection missions is a challenging and safety-critical task. As with conventional methods for Earth observation scheduling, applying motion planning techniques to the inspection problem can yield brittle plans at high computational costs. Because many cases feature uncertainty in the geometry and dynamics of the RSO, the problem benefits from being performed in a closed-loop manner without ground intervention. Again, these factors necessitate the development of increased autonomy capabilities.

Motivated by the need for increased autonomy capabilities, this dissertation proposes methods for policy-based autonomy rather than plan-based execution for spacecraft operations. To this end, this work advances the application of RL to these classes of space-based observation tasks. To do so, RL-based approaches are developed and compared to conventional planning methods in realistic instances of the problems, featuring the constraints and requirements characteristic of aerospace systems. Across the considered applications, RL is demonstrated as an effective way of achieving closed-loop autonomy for realistic and relevant space missions.

1.2 Prior Work

This section gives a general overview of conventional approaches to planning and scheduling for aerospace systems and compares them to the state of the art for applying RL to these classes of problems. A brief development of concepts and vocabulary from RL that form the basis of the field is also included.

1.2.1 Reinforcement Learning

Sequential decision-making encompasses a broad class of problems ranging from motion planning and games to optimal control and filtering. As such, RL has wide applicability and utility, providing a pathway for solving such problems through iterative interaction when formulating as Markov decision processes (MDPs). RL can be traced to the fields of optimal control and dynamic

programming, developed in the 1950's by Bellman and others, and trial-and-error learning for animals, featured in the early 20$^{\text{th}}$ century work of Pavlov. While Turing first identified the potential for an electromechanical "pleasure-pain system" in 1948, it was not until the work of Klopf, Sutton, and Barto in the 1970s and 80s that the threads rigorously merged into what is now known as RL [25].

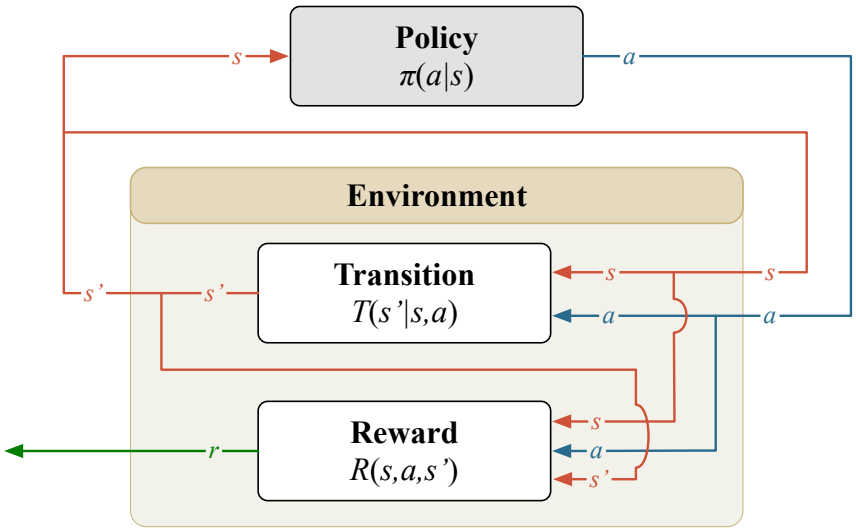


Figure 1.2: The basic MDP interaction framework.

The MDP 4-tuple (Figure 1.2) has a state space $s \in \mathcal{S}$, action space $a \in \mathcal{A}$, transition probability distribution $T(s'|s,a)$ which describes the chance of transitioning from state s to state s' when selecting action a, and per-step reward function $R(s,a,s')$ which describes the benefit or penalty for transitioning to a certain state. To solve the MDP, one must find a policy $\pi(a|s)$ that describes which action should be selected in each state in order to maximize the reward signal over all steps. Because finite rewards may appear at infinitely many steps in the future, a discount factor $\gamma \in [0,1]$ is introduced to prevent an infinite summation, prioritizing near-term rewards and making rewards in the far future valueless. Formally, the expected sum of future rewards under a policy π is known as the value

$$V^{\pi}(s) = r + \gamma \mathbb{E}\left[V^{\pi}(s') \middle| T(s'|s,a), \pi(a|s)\right] \tag{1.1}$$

which for a known trajectory through the MDP can be computed as

$$= r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = \sum_{i=0}^{\infty} \gamma^i r_i \tag{1.2}$$

Thus, the problem is an optimization over the space of policies:

$$\underset{\pi}{\text{maximize}}\ V^{\pi}(s) \quad \forall\ s \in \mathcal{S} \tag{1.3}$$

Variants of the MDP are defined with additional components to better express certain types of problems:

- The semi-Markov decision process (sMDP) introduces a notion of continuous time to MDPs, rather than only having discrete steps [26, 27, 28]. Each step has some Δt associated with it, that—when discounting—can be used as a time opportunity cost instead of a step opportunity cost. The most general form replaces the transition distribution T with a joint distribution $Pr(s', \Delta t|s, a)$ and the scalar reward r with a reward density $\varrho(t)$. A treatment of the mathematical impacts of the sMDPs when training is given in subsection 5.2.1.

- In a partially-observable Markov decision process (POMDP), the true state s is not exposed. Instead, an observation probability distribution $O(o|s)$ is defined to map states s to observations $o \in \mathcal{O}$. In these formulations, policies respond to observations instead of states: $\pi(a|o)$. Partial observability can be used to represent noisy measurements, uncertainty, or as a method of removing irrelevant information from the state. Depending on the problem, it can be beneficial to include a filter-like component prior to the policy to estimate the true state before acting.

- Decentralized partially-observable Markov decision processes (Dec-POMDPs) allow for problems with multiple independently-acting agents to be represented. Instead of a single action space $\mathcal{A}$, each agent i selects their action from a joint action space $\{a_i \in \mathcal{A}_i\}$. Likewise, agents act on observations $\{o_i \in \mathcal{O}_i\}$ generated from a joint observation probability distribution $O(\{o_i\}|s)$. Rewards are also yielded on a per-agent basis: $R(s, \{a_i\}, s') \to \{r_i\}$. Decentralized problems can be collaborative [29] or adversarial.

Algorithm 1 PPO

Initialize policy parameters θ_0 for policy π_θ and value function parameters ϕ_0 for value function estimator $\tilde{V}_\phi$
for $k = 1, 2, 3, ...$ **do**
Collect trajectories by executing policy π_{θ_k} in the environment
Estimate advantages $A^{\pi_{\theta_k}}(s, a)$ for each step in trajectories using $\tilde{V}_\phi$ with GAE
$\theta_{k+1} \leftarrow$ Update the policy parameterization by maximizing PPO-Clip objective (Equation 1.4) using the Adam optimizer [35]
$\phi_{k+1} \leftarrow$ Update the value function parameterization by minimizing the error $\left(\tilde{V}_\phi(s_t) - \sum_{\tau=t}^{t_f} \gamma^{\tau-t} r_\tau\right)^2$ across collected trajectories
end for

In RL, an agent iteratively learns a policy with which to interact with an environment described by an MDP in such a way that rewards are obtained and penalties are avoided. However, only the state space $\mathcal{S}$ and action space $\mathcal{A}$ are known to the agent; no prior knowledge of the transition function T or reward function R is provided; instead, the agent gains experience about how to reach high-reward states through interaction. A key challenge for these algorithms is balancing exploration, or searching new areas of the state space, with exploitation, or revisiting areas of the state space with known benefits.

Deep reinforcement learning (DRL), which uses neural networks (NNs) to approximate functions within RL algorithms, has been studied since the early days of the field. DRL can provide strong performance on problems that are intractable for exact methods and benefits from the generalization properties of NNs. Recently, advances in algorithms and computing hardware have led to a renaissance in deep learning. These accomplishments include creating grandmaster-level agents for board games such as chess and go, simple video games like the Atari catalog [30] and highly complex video games such as StarCraft [31]. The sim-to-real gap has been bridged in various cases, applying RL to problem settings such as dexterous robotic hand manipulation of objects [32] and sample-efficient quadrupedal robotic control [33]. More broadly, the proliferation of accessible implementations of algorithms such as proximal policy optimization (PPO) that perform well across domains has enabled the application of RL to become widespread across fields [34].

PPO is a policy gradient method, meaning that it directly optimizes the policy π_θ by itera-

tively updating the policy parameters θ as the agent gains experience in the environment. Pseudocode summarizing how PPO works is given in Algorithm 1.[1] To prevent unlearning, PPO limits the policy update to a small region around the current policy using the loss function

$$L\left(s, a, \theta_k, \theta\right) = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s, a), \quad g\left(\epsilon, A^{\pi_{\theta_k}}(s, a)\right)\right) \tag{1.4}$$

where the clipping function g is

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases} \tag{1.5}$$

Advantage A^π for a policy π is defined how valuable an alternative action is relative to the value of the policy's desired action

$$A^\pi(s, a) = \mathbb{E}\left[R(s, a) + V^\pi(s')\right] - V^\pi(s) \tag{1.6}$$

Generalized advantage estimation (GAE) is often used for computing advantage in PPO. The standard form of the GAE

$$\hat{A}^{\text{GAE}}_{t,\text{MDP}} = \sum_{i=0}^{\infty} \lambda^i \gamma^i \left(r_i + \gamma V(s_{t+1}) - V(s_t)\right) \tag{1.7}$$

which takes λ-decaying average of advantage estimates, was introduced by Schulman et al. [36].

1.2.2 Limitation of Conventional Planners

Many of the conventional methods used for spacecraft planning and scheduling have two key limitations when compared to distributed, autonomous systems:

(1) They use open-loop, ground-based planners that are subject to communication delays and brittle to changing conditions.

(2) They plan over expert-designed abstractions of the problem space that may not completely capture true system dynamics [6].

[1] Adapted from https://spinningup.openai.com/en/latest/algorithms/ppo.html#pseudocode.

Such planners use a variety of common combinatorial optimization algorithms for task scheduling, such as MILP, ILS, constraint programming, and genetic methods [37, 38, 39, 40, 41, 42], and path-planning algorithms, such as probabilistic roadmaps (PRMs) and A* [43, 44]. These methods work by generating a mathematical program representation of the problem with a combination of discrete and continuous constrained variables based on an expert understanding of the problem, then solving them with an exact or heuristic solver. For the AEOSSP, the problem is often reduced to a form of the traveling salesman problem (TSP) over a directed acyclic graph (DAG) of requests and transitions [45]. Often, these formulations place restrictions on the types of dynamics that can be modelled: Coupled and nonlinear effects such as power generation as a function of attitude or fuel consumption must be linearized, approximated, or neglected [38, 7]. Transition times between tasks, such as slew maneuvers, are approximated by simplified models that can produce infeasible or suboptimal plans [46]. Simplified dynamics with closed-form solutions, like Clohessy-Wiltshire-Hill (CWH) approximations, are necessary for some planning methods. Often, methods require the discretization of continuous spaces into waypoints [47] or fixed planning intervals [48], both of which result in suboptimal plans. The size of the resulting optimization problems tends to grow combinatorially with the number of spacecraft and tasks, leading to computationally-prohibitive problems or suboptimal solutions due to a limited solution time. The resulting plans are not responsive to the actual system dynamics, so unexpected performance or opportunistic events may depress the performance of the plan. In some cases, uncertainty is accounted for in the deterministic plan [49, 50]. There has been some research into onboard plan repair by reoptimizing over a short horizon, but the cost of common solvers can easily exceed onboard capabilities [3]. Much of the work in this domain is applied to Earth observation, in which a plan is made or modified in a closed-loop response to images processed onboard [11, 16, 51]. Finally, certain scenarios may have uncooperative or even adversarial elements that must be responded to in a closed-loop manner [52].

MILPs and other offline optimizers do offer certain advantages over RL; namely, they can encode guarantees about properties of the plan. These may take the form of safety requirements (including keep-ins, keep-outs, and limit on power, fuel, or time) or contractual obligations (such as

service-level agreements for Eath-observing constellations) that an optimizer can strictly enforce. While DRL can be encouraged to act in a way that is compliant with these constraints, additional techniques must be employed to guarantee performance. For continuous control, control barrier functions (CBFs) [53] can enforce guarantees such as keep-ins, as used in Van Wijk's autonomous inspection work [54]. Shields [55] can be developed to provide guarantees on avoiding unsafe regions in discretely controlled environments, as done for a satellite tasking spacecraft in Reed's work [56, 57]. However, such methods for RL are still an open area of research that cannot yet strictly enforce the range of performance requirements encodable in offline optimizers.

1.2.3 Learned Autonomy in Aerospace

Researchers in aerospace have begun to acknowledge that the breakthroughs enabled by RL and DRL in other domains may be achieved for space-based observation tasks, offer a promising alternative that addresses the challenges of conventional methods. Agents learn in a simulation of the satellite tasking environment, allowing for performance to be maximized with respect to arbitrarily complex models of the system's dynamics, constraints, and objectives. The resulting closed-loop policy can be deployed on the spacecraft, allowing it to react in real-time to new information or unexpected events (e.g. faults or opportunistic events) that it encounters. When represented by a multilayer perceptron (MLP), the policy is computationally low-cost system; more exotic architectures may incur higher computational costs but can still be efficiently evaluated on appropriate hardware. Table 1.1 summarizes how RL as a policy-based method compares to traditional plan-based methods and plan-based methods with some onboard decision making. Research applying RL to spacecraft control and tasking problems has proliferated in the past seven years. The first formulations of satellite tasking as varieties of MDPs appear in works by Harris [58], Eddy [28], and Hadj-Salah [59], all applied to simplified formulations of the AEOSSP.

This research has expanded into RL algorithms that are fine-tuned for particular parts of the AEOSSP [60, 61, 62], challenges associated with applying RL in a more flight-like scenario [63], and various treatments of the multi-agent scheduling problem [64, 65, 66]. Some work has enhanced the

Table 1.1: Comparison of costs, requirements, and capabilities of RL and traditional planning frameworks.

	Ground Only (MILP, ILS, etc.)	**Ground + Onboard Reasoning**	**RL**
Approach	Plan-based	Plans with decision logic	Policy-based
Modeling Difficulty	Moderate, requires expertise	High, two types of planners	**Lower, requires simulator**
Modeling Capabilities	Method- and problem-dependent	Method- and problem-dependent	**Anything simulatable**
Algorithm Complexity	High, but tools exist	High, but tools exist	High, but tools exist
Reactivity/Robustness	Open-loop; limited	Closed-loop for anticipated choices	**Closed-loop**
Constraint Satisfaction	**Strong**	Requires compliance at two stages	Requires extra analysis
Compute (Mission Design)	**Low**	**Low**	High (training costs)
Compute (Ground Segment)	High	High	**None; onboard planning**
Ground Communication	Required	Required	**Unnecessary**
Compute (Onboard)	**Low; plan execution**	Moderate; decision logic	**Low; policy evaluation**

traditional methods with RL while still maintaining the basic graph construction pipeline [60, 61]. Ultimately, these methods are all subject to preplanning's limitations, in that they are brittle to changing request lists or environments. Nazari identified that orienteering problems, of which the AEOSSP is an instance, are solvable by RL [67]; soon after, RL was applied to solve simplified versions of the spacecraft tasking problem [68, 59, 28]. Recent studies have focused on adding resource management and safety requirements to the problem that cannot be easily handled in the optimization-based frameworks, but have maintained simplifications on task durations that prevent direct comparisons with the preplanning approach [58, 63, 69]. This thread of research has been enhanced by the automatic generation of provably safe shields for these systems, which is critical to adopt these methods operationally [58, 70, 56].

While interest in the RSO inspection problem is more recent, researchers are considering diverse solution methods, many of which leverage RL. The earliest relevant work is on satellite swarm reconfiguration with continuous control [71, 72] and impulsive control [73, 74]. Unlike prior work on low-level relative orbit control [75], these reconfiguration methods incorporate concepts from multi-agent path planning. When applied to the inspection problem, approaches for trajectory generation include linear programming [76], covering orbit generation and transfer optimization [43, 77], and other bespoke methods [44].

RL is used in recent treatments of the problem. Considering safety, MDPs have been applied for safe maneuver planning [78]. For inspection, several authors have taken discrete waypoint-based approaches [79, 80, 81, 82]. Other authors have considered the problem with continuous low-thrust control, focusing on shape reconstruction [83, 84] and transformer-based rendezvous [85, 86]. Most similar to the problem and methods presented in this work are the papers of Van Wijk and Dunlap [54, 87], who pose a RL-based solution to the inspection task with continuous control and control barrier functions to guarantee safety [53]. With the limited flight heritage of any methods in RSO inspection, RL could establish itself as a method of choice early in the technology's lifecycle.

Various problems adjacent to the RSO inspection problem have also been considered in autonomy literature. Relative motion control for inspection of small bodies [88, 89, 54] and rendezvous

and docking [90, 91] have emerged as popular continuous control applications of RL in the space domain. It has has also been utilized for orbital control, with applications in orbital rendezvous [92] and interplanetary [93] and cislunar [94, 95] trajectories.

Even among these examples of applications of RL, many use simplified models of the system dynamics for task transitions, task completion, and resource constraints. Examples include the assumption of fixed task durations [58], CWH dynamics [54], linear models for power consumption and data capacity [28], and simple slew time dynamics [64]. While these simplifications make apples-to-apples comparisons with planning and control methods that require such assumptions easier, they do not fully leverage the capabilities of DRL to learn on arbitrarily complex environments that capture more real-world nuances. While genetic methods and similar offline optimizers can be used to solve these MDP-formulated problems, they tend to be highly inefficient, requiring many simulations to optimize a single case; additionally, they lack both the closed-loop responsiveness of RL and the optimality and constraint guarantees of other offline optimization frameworks.

1.3 Summary of Contributions

The goal of this thesis is to demonstrate that policy-based autonomy via RL can safely and performantly operate autonomous spacecraft across a variety of challenging and mission-realistic tasks. First, RL-based policies are shown to perform competitively with traditional optimization-based methods for the single-agent Earth observation, while providing additional benefits of closed-loop operation; for this, tools and optimal benchmarks must also be developed. From there, the methods are extended to multi-agent Earth-observation problems that are challenging or impossible for traditional optimization methods. Finally, the lessons learned from the single-agent and multi-agent Earth observation tasks are transferred to the problem of autonomous RSO inspection with RL and motion planning algorithms; this problem presents the challenges of complex dynamics and continuous action spaces, but similar fundamental considerations of performance, safety, and multi-agent effects are present. In support of all research goals, the BSK-RL framework is designed and implemented to provide a high-fidelity simulation of general spacecraft tasking problems with

a MDP interface (chapter 2). The dissertation can be broadly categorized into three focuses:

- **Focus I: Single-Satellite Earth Observation Scheduling** The first research focus looks at the AEOSSP (chapter 3) for a single satellite. In particular, it demonstrates that RL can perform competitively with ground-based preplanning approaches to request sequencing under realistic resource-constrained conditions, and it also shows that RL enables additional capabilities beyond the limitations of current planners. This focus considers two methods: First, traditional MILP-based approaches to satellite scheduling are adapted for use with the high-fidelity simulation in order to provide an optimal and practical baseline for comparison (chapter 4). Next, the problem is solved with DRL using a sMDP formulation to account for variable-duration actions (chapter 5). The resulting policy is competitive with the baseline provided by the MILP solver; constraints and challenges are added to the problem formulation to demonstrate how RL handles cases that traditional techniques can not.

- **Focus II: Asynchronous Collaboration for Observation Constellations** Building on the single-satellite work of the first focus, the second focus investigates the agile Earth-observing constellation scheduling problem (AEOCSP) considering constellations of agents and scheduling tasks beyond those with *a priori* task lists. In a multi-agent version of the AEOSSP, the responsive capabilities of the single-agent policies are leveraged to induce collaboration in a multi-agent setting via task deconfliction algorithms (chapter 6); as with the single-agent task in the first focus, the multi-agent method performs near-optimally when quantified against an optimal solver. To demonstrate a mission architecture enabled by RL, a distributed tip-and-cue problem is formulated and solved using multi-agent reinforcement learning (MARL) (chapter 7). This task is challenging because it requires agents to demonstrate closed-loop responsiveness and actively collaborate in order to perform well.

- **Focus III: Safe Inspection of Space Objects** The final focus investigates a different observation problem than the previous two, namely, autonomous inspection of a RSO

leveraging relative motion for fuel efficiency with impulse maneuvers for control (chapter 8). First, the problem is solved with RL, considering illumination constraints, elliptical orbits, and variable fuel cost (chapter 9). Given the risks associated with maneuvering around another object, analytical, optimization-based shields are derived to guarantee safe maneuvers, preventing the policy from selecting actions that would lead to collision between the servicers and RSO. As with the progression from Focus I to II, the single-agent RL formulation and shield is extended to a multi-agent case with both the single-agent policy and a MARL-based policy. To quantify the performance of the RL formulation, a PRM-based planner is developed for the inspection problem (chapter 10); with it, the trade-offs between the two methods across the fuel-time Pareto front can be analyzed.

1.4 Publications

All publications and presentations that originated during this Ph.D. work are listed below. For journal publications, the chapters of the dissertation they were developed into are identified.

1.4.1 Journal Papers

(6) **M. Stephenson** and H. Schaub, "Autonomous Tip-and-Cue Earth-Observing Constellation Tasking with Reinforcement Learning," *IEEE Transactions on Aerospace and Electronic Systems*. Submitted April 2026. [Chapter 7]

(5) **M. Stephenson** and H. Schaub, "Two Approaches to Spacecraft Inspection: Comparing Shielded Reinforcement Learning and Probabilistic Roadmaps," *AAS Journal of the Astronautical Sciences*. Submitted January 2026. [Chapters 8–10]

(4) **M. Stephenson**, L. Quevedo Mantovani, and H. Schaub, "Inducing Collaboration Between RL-Based Agents in a Distributed Constellation," *AIAA Journal of Aerospace Information Systems*. Submitted December 2025. [Chapter 6]

(3) **M. Stephenson**, L. Quevedo Mantovani, and H. Schaub, "Learning Policies for Au-

tonomous Earth-Observing Satellite Scheduling over Semi-MDPs," *AIAA Journal of Aerospace Information Systems*, Vol. 22, No. 9, September 2025, pp. 741–803. doi:10.2514/1.I011649. [Chapters 3, 5]

(2) **M. Stephenson** and H. Schaub, "Optimal Agile Satellite Target Scheduling with Learned Dynamics," *AIAA Journal of Spacecraft and Rockets*, Vol. 62, No. 3, May–June, 2025, pp. 793–804. doi:10.2514/1.A36097. [Chapters 3, 4]

(1) A. Herrmann, **M. Stephenson**, and H. Schaub, "Single-Agent Reinforcement Learning for Scalable Earth-Observing Satellite Constellation Operations," *AIAA Journal of Spacecraft and Rockets*, Vol. 61, No. 1, January–February 2024, pp. 114–132. doi:10.2514/1.A35736.

1.4.2 Technical Notes

(1) **M. Stephenson** and H. Schaub, "BSK-RL: Modular, High-Fidelity Reinforcement Learning Environments for Spacecraft Tasking," *AIAA Journal of Aerospace Information Systems*. Submitted January 2026. [Chapter 2]

1.4.3 Conference Papers

(12) **M. Stephenson** and H. Schaub, "Autonomous Tip-and-Cue Earth-Observing Constellation Tasking with Reinforcement Learning," IEEE Aerospace Conference, Big Sky, Montana, March 7–14, 2026.

(11) **M. Stephenson** and H. Schaub, "Comparing Probabilistic Roadmaps and Reinforcement Learning for Relative Motion Inspection of Space Objects," AAS Rocky Mountain GN&C Conference, Breckenridge, Colorado, January 31–February 4, 2026.

(10) **M. Stephenson** and H. Schaub, "Safe, Autonomous Multi-agent Inspection of Space Objects Leveraging Relative Orbit Dynamics," Advanced Maui Optical and Space Surveillance Technologies Conference, Maui, Hawaii, September 16–19, 2025.

(9) **M. Stephenson**, D. Huterer Prats, and H. Schaub, "Autonomous Satellite Inspection in Low Earth Orbit with Optimization-Based Safety Guarantees," International Workshop on Planning and Scheduling for Space, Toulouse, France, April 28–30, 2025.

(8) **M. Stephenson**, L. Quevedo Mantovani, A. Cheval, and H. Schaub, "Quantifying The Optimality Of A Distributed RL-Based Autonomous Earth-Observing Constellation," AAS Rocky Mountain GN&C Conference, Breckenridge, Colorado, January 31–February 5, 2025.

(7) **M. Stephenson** and H. Schaub, "Scalable Autonomous Decentralized Constellation Tasking on Asynchronous Semi-Markov Decision Processes," International Workshop on Satellite Constellations & Formation Flying, Kaohsiung, Taiwan, December 2–4, 2024.

(6) **M. Stephenson** and H. Schaub, "BSK-RL: Modular, High-Fidelity Reinforcement Learning Environments for Spacecraft Tasking," International Astronautical Congress, Milan, Italy, October 14–18 2024.

(5) **M. Stephenson**, L. Quevedo Mantovani, and H. Schaub, "Intent Sharing For Emergent Collaboration In Autonomous Earth Observing Constellations," AAS Rocky Mountain GN&C Conference, Breckenridge, Colorado, February 2–7, 2024.

(4) **M. Stephenson** and H. Schaub, "Reinforcement Learning for Earth-Observing Satellite Autonomy with Event-Based Task Intervals," AAS Rocky Mountain GN&C Conference, Breckenridge, Colorado, February 2–7, 2024.

(3) **M. Stephenson**, L. Quevedo Mantovani, S. Phillips, and H. Schaub, "Using Enhanced Simulation Environments to Accelerate Reinforcement Learning for Long-Duration Satellite Autonomy," AIAA Science and Technology Forum and Exposition (SciTech), Orlando, Florida, January 8–12, 2024. doi:10.2514/6.2024-0990.

(2) **M. Stephenson** and H. Schaub, "Optimal Target Sequencing In The Agile Earth-Observing Satellite Scheduling Problem Using Learned Dynamics," AAS/AIAA Astrodynamics Specialist Conference, Big Sky, Montana, August 13–17 2023.

(1) A. Herrmann, **M. Stephenson**, and H. Schaub, "Reinforcement Learning for Multi-Satellite Agile Earth Observing Scheduling Under Various Communication Assumptions," AAS Rocky Mountain GN&C Conference, Breckenridge, Colorado, February 2–8, 2023.

1.4.4 Conference Presentations

(2) **M. Stephenson** and H. Schaub, "Safe, Autonomous Multi-Agent Inspection of Space Objects Leveraging Relative Orbit Dynamics," 4th Annual AI for Contested Space Workshop, Lexington, Massachusetts, September 23–25, 2025. Presentation only.

(1) **M. Stephenson** and H. Schaub, "Achieving Near-Optimal Performance in Autonomous Earth Observing Satellite Scheduling using semi-MDPs," International Workshop on Planning and Scheduling for Space, Toulouse, France, April 28–30, 2025. Presentation only.

Chapter 2

BSK-RL: Environments for Spacecraft Tasking

2.1 Motivation

The standardization of reinforcement learning (RL) environments—i.e. interactive generative Markov decision process (MDP) simulators—has been of key importance for improving the quality, efficiency, and reproducibility of research in the field of RL as a whole [108]. Widespread adoption of the Python-based Gymnasium (formerly Gym, developed by OpenAI) environment API and PettingZoo multi-agent API has led to a de facto standard for interfacing RL algorithms with environments [109, 110]. These APIs provide standardized ways of accessing actions, observations, and rewards and of stepping and resetting the learning environment. Specific environments implemented using these APIs have also emerged as standard benchmarks for RL algorithms: Atari games like Breakout, Tetris, and Adventure have become key benchmarks for image-input, discrete control environments [111, 112]. The MuJoCo physics engine has Gymnasium interfaces that can be used to create a variety of continuous control environments like Ant, Hopper, and Half-Cheetah, as well as arbitrarily complex multibody physics-based environments [113]. Gymnasium interfaces have been developed for modern video games such as Minecraft [114] and Starcraft II [31], providing challenging benchmarks for RL algorithms in complex, high-dimensional, and partially observable environments. While these software packages provide a diversity of open-source environments (or open-source interfaces into closed-source software) for RL research, they are primarily seen as a testing ground for algorithms; the environments themselves are not the focus of research. MuJoCo is an exception, as arbitrary physics-based environments of interest can be created within the engine

and interfaced with Gymnasium.

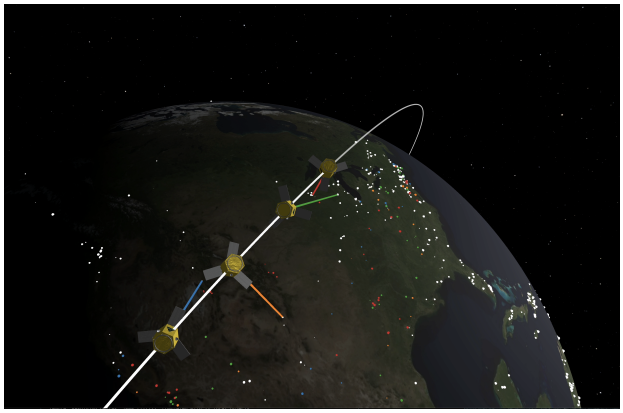
Within the space domain, open-source RL environments are lacking; existing options are compared in Table 2.1. "KSPDG: Kerbal Space Program Differential Games" implements various non-cooperative spacecraft control tasks using the Gymnasium API and the video game Kerbal Space Program (KSP) as a physics engine [52]. This environment is designed primarily as a real-time evaluation environment of policies and control methods developed externally to the environment, encouraging users to treat KSPDG as "reality" when trying to overcome the sim-to-real gap. As a result, it is self-described as not appropriate for training agents due to the high computational overheads and real-time simulation of the KSP backend. RANS and Space Robotics Bench implement a variety of space robotics tasks using the NVIDIA Isaac Sim framework, which is optimized for RL [115, 116]. While these packages have some overlap with BSK-RL's capabilities for simulating rendezvous tasks, BSK-RL focuses on higher-level spacecraft planning and scheduling as opposed to lower-level robotics activities; accordingly, it models both the dynamics and flight software (FSW) of agents. While other aforementioned spacecraft RL research has defined and implemented environments for particular problems, they tend to be low-fidelity, problem-specific models that lack a maintained, open-source repository for general use.

The packages listed in Table 2.1 cover many of the most-used dynamics engines for RL research [120]. Some have associated space-application-specific wrappers, but Gazebo and MuJoCo do not have native or third-party support for space environments; additional work would be required to implement the necessary dynamics models for a space application. These robotic-focused simulation frameworks tend to focus on actuator-level control and do not natively support the simulation of algorithms running onboard the robot. While this allows for GPU-accelerated dynamics simulation, it restricts the behaviors and problems that can be modelled, including task-based commanding that is of interest in this work. Those simulators with FSW-like algorithm execution are identified in the table, but this does not necessarily indicate realistic FSW stack simulation.

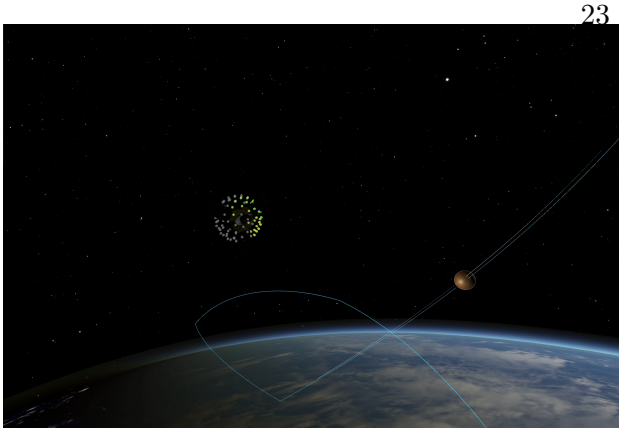
BSK-RL aims to fill this gap by providing a high-fidelity, high-speed, open-source, and mod-

Table 2.1: Comparison of packages for space and general robotics RL applications.

Space Package	Simulator	Dynamics	Focus	Speed	Extensible	FSW	Multi-agent	Cite
BSK-RL	Basilisk	BSM or MuJoCo	Orbital tasking and control	Fast	Yes	Yes	Yes	[117]
Space Robotics Bench	Isaac Sim	Newton	Mobile and manipulation space robotics	Fast	Yes	No	No	[116]
General use	Isaac Sim	Newton	General robotics	Fast	Yes	No	Yes	[118]
SpaceGym	KSP	Unity	Adversarial orbital tasks	Slow	No	Yes	Yes	[52]
General use	Gazebo	ODE	General robotics	Medium	Yes	ROS	Yes	[119]
General use	MuJoCo	MuJoCo	General robotics	Fast	Yes	No	Yes	[113]



(a) Earth observation task.[a]



(b) Satellite inspection task.[a]

[a] https://www.youtube.com/watch?v=lCN0TiNJ1i4

[a] https://www.youtube.com/watch?v=eQEoTOYADKc

Figure 2.1: Examples of BSK-RL scenarios.

ular environment for spacecraft tasking problems with an open-source repository[1] , user-friendly documentation[2] , and example scripts. At its core, BSK-RL combines Basilisk[3] [117], a spacecraft dynamics simulation package, with abstract mission objectives, wrapped together in the Gymnasium and PettingZoo APIs. The package is designed to meet the needs of RL and spacecraft operations researchers: Environment parameters are easily reproducible, customizable, and randomizable. Environments are highly modular: satellite observation and action spaces—both discrete and continuous—can be specified, mission objectives and rewards can be defined, and the satellite dynamics and FSW can be configured, implicitly introducing operational limitations and safety constraints. The framework allows for the creation of a wide range of complex mission scenarios (Figure 2.1) featuring heterogeneous multi-agent environments that consider communication and collaboration.

This work presents two key contributions to the field: 1) a generalized framework for spacecraft tasking RL environments that can be modified and extended; and 2) well-defined environments with benchmarks that can be used to compare algorithms when advancing the state of the art for

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

spacecraft applications. Motivated by the existence of common features and architectures across spacecraft planning environments, well-tested, flexible frameworks for those components streamline the creation of new environments and the extension of existing environments. The complete environments defined within BSK-RL support the development of training algorithms for orbital scenarios by establishing a common benchmark for evaluation. This work describes the design and capabilities of BSK-RL in order to explain the range of scenarios that it can be used to study. Scenarios featured in existing literature, including proximity operations and Earth-observing constellations, are provided and benchmarked with proximal policy optimization (PPO) to offer the community a reference for further algorithm development.

2.2 Dependencies

BSK-RL is built on open-source dependencies including Basilisk, a spacecraft simulation framework that forms the base simulation of the environments, and Gymnasium and PettingZoo, standard APIs for RL environments that yield compatibility with external tools. BSK-RL bridges these two fronts.

2.2.1 Basilisk

Basilisk is a modular spacecraft simulation framework written in C and C++ with a Python interface [121, 117]. The package is capable of simulating spacecraft subject to multibody dynamics (including reaction wheels and actuated solar arrays), environmental effects (such as planetary gravitational forces and atmospheric drag), power and data storage subsystems, and onboard FSW with actuator-level control. The architecture enables relatively complex configurations to achieve simulation speeds hundreds of times faster than real time using the Backsubstitution method (BSM) for rigid body dynamics, which decomposes the mass matrix inversion into cheaper operations based on the spacecraft topology [121]. This combination of speed and configurability makes Basilisk an attractive environment for RL. Basilisk has been validated in many ways. Rigid body dynamics components have been analytically validated [122, 123, 124, 125]. Full dynamics simulations and

Table 2.2: Default BSK-RL environmental models and simulator properties from Basilisk.

Effect	Default Model	Notes	Cite
Earth Gravity	Grace Gravity Model 03 (GGM03)	spherical harmonic, degree 10	[126]
Celestial Mechanics	SPICE Toolkit	Earth and Sun position	[127]
Atmospheric Drag	exponential atmosphere, faceted drag	produces force and torque	
Integrator	Runge-Kutta 4th order	fixed-step integrator	
Rigid Body Dynamics	BSM	efficient, acceleration-based	[121]

FSW algorithms have been validated for use with flight missions such as the Emirates Mars Mission. Alternative commercial options (STK, FreeFlyer) suffer from the usual drawbacks of closed-source software: large computational overheads, limited APIs, high costs, and a lack of extensibility.

For a sense of the fidelity of the simulation, Table 2.2 lists environmental models and other simulator properties from Basilisk that BSK-RL uses by default. Models of all common spacecraft components are supported within the rigid body dynamics framework. Since both Basilisk and BSK-RL are highly modular, alternative models of higher or lower fidelity implemented in Basilisk can easily be substituted. As alternatives to the defaults used by BSK-RL, Basilisk has implementations of the MSIS atmosphere model [128], RK1-4, RKF45, RKF78, and Bogacki-Shampine integrators, and a MuJoCo dynamics backend for physical architectures not compatible with the BSM [113].

In BSK-RL, Basilisk serves as a truth model for the environments that is validated externally from BSK-RL. Once configured by BSK-RL, the Basilisk simulation runs independently over the course of a step until the process returns to BSK-RL to process the results, such as each satellite's physical and FSW states. As a result, the development focuses of BSK-RL are the configuration and interpretation of the simulated spacecraft, and not the dynamics simulation itself since independently developed and verified physics and FSW models from Basilisk are being used. Thus, the spacecraft activities that are executed in the RL environments are being executed directly in a simulation environment that is of appropriate accuracy and fidelity to use for real spacecraft

operations.

2.2.2 Gymnasium and PettingZoo

In order to leverage the plethora of existing RL libraries in Python, BSK-RL exposes the learning environment via the Gymnasium [109] and PettingZoo [110] APIs. Gymnasium is the standard interface for defining single-agent MDPs in Python. All major RL libraries, such as RLlib [129], and Stable Baselines [130] are compatible with the Gymnasium API. PettingZoo extends the Gymnasium API to handle cooperative and competitive multi-agent scenarios.

Gymnasium environments are generative models of the partially-observable Markov decision process (POMDP) formalism. A POMDP is defined by the state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability distribution $T(s'|s,a)$, reward function $R(s,a,s')$, observation space $\mathcal{O}$, and observation probability distribution $Z(o|s)$. Many of these distributions are difficult to compute explicitly. For RL, only a generative model $o, r, s' = G(s,a)$ that propagates the environment given an action and returns an observation of the new state and associated reward is necessary; in effect, a not-necessarily-deterministic simulator. Gymnasium defines a framework for this kind of generative, simulator-based POMDP interaction in Python. While these environments are best described as POMDPs due to the hidden, complete simulator state determining how the environment evolves, learning generally occurs only on the observation without explicit estimation of the underlying state. If the observation includes all relevant information about the problem, this approach is reasonable.

Gymnasium environments implement two main functions: The reset function `obs, info = env.reset(seed)` sets up the initial state of an environment and handles environment condition randomization. Environment randomization is a function of the seed, so particular environment instantiaions can be reproduced by using the same seed. The step function `obs, reward, term, trunc, info = env.step(action)` provides the fundamental means of interaction between an agent and its environment: The environment is updated depending on the action selected by the agent, and the new state and reward are returned (along with information about the environment's sta-

tus, including terminated and truncated episode flags). To supply the environment interface to the learning agent, the environment implements the `observation_space` and `action_space` properties, respectively defining the domains for the observations the agent expects to receive and the actions (discrete or continuous) that it can take. In BSK-RL, the base single-agent environment is the `SatelliteTasking` environment. For multi-agent scenarios, BSK-RL adopts the PettingZoo Parallel API for the `ConstellationTasking` environment. With this API, environments process observations, actions, and rewards for multiple agents that interact with the same environment in parallel. Not all agents are expected to act at every step, and agents may fail at different times. This is reflective of distributed spacecraft operations, in which each spacecraft is often independently and asynchronously controlling itself.

2.3 Architecture

BSK-RL's architecture is organized into three areas:

(1) the underlying Basilisk simulation, which gives the physical behavior of the satellites in the environment;

(2) the satellite agents that act in the environment, with configurable interfaces for observations and actions;

(3) the optimization objective of the environment, which represents mission goals and constraints

Working together, the environment connects the dynamics of the high-fidelity simulation to abstract tasks and goals that must be achieved. By having control of the Basilisk simulation occur via command and configuration of the satellites' FSW models, the ground truth provided by the simulation evolves realistically. Multi-agent capabilities are built-in, allowing for complex, constellation-based scenarios to be created with communication between agents.

Figure 2.2 demonstrates how information flows through components of the environment, which are described in the following subsections. Actions from the Gymnasium API are processed by

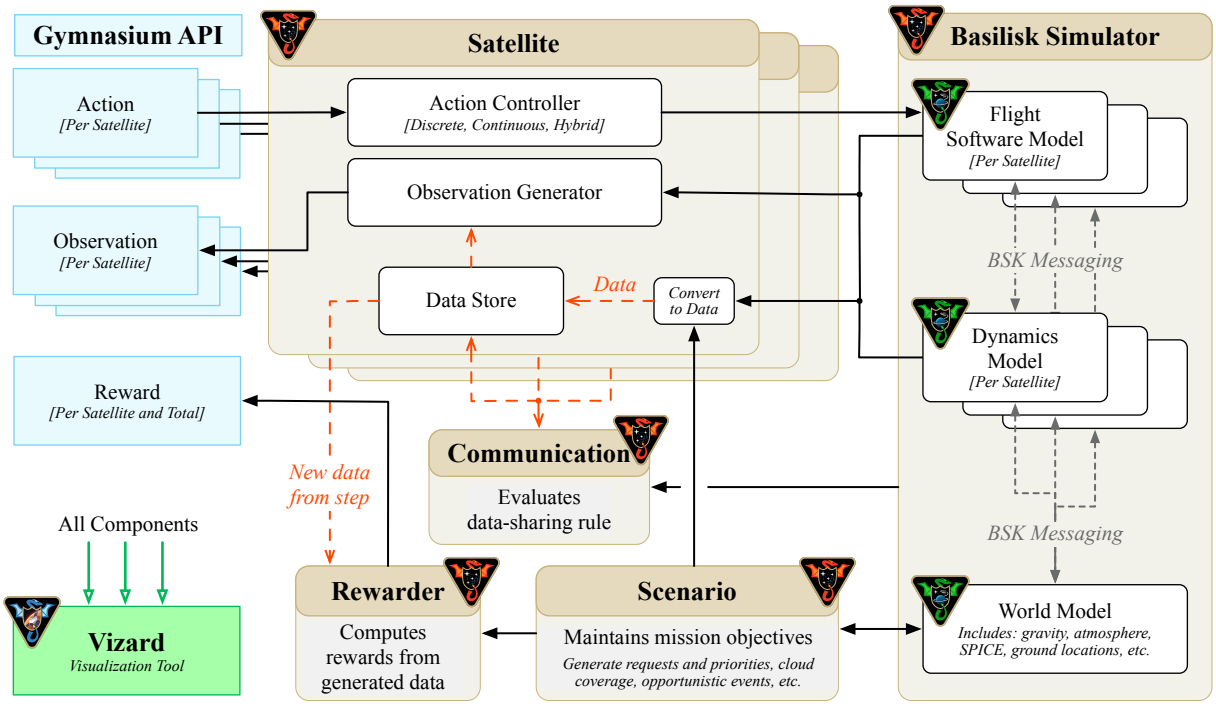


Figure 2.2: The architecture of information flow between components in the BSK-RL environment.

each satellite, changing the configuration of the simulated FSW. The underlying Basilisk simulation executes until an action completes or a termination condition is met. The results of the Basilisk simulation are processed by various environment components. Based on the state of the simulation, data—a representation of progress toward some abstract goal—is generated by each satellite. This data is shared between satellites according to communication rules and is used to compute rewards. The simulator and data are also processed to generate per-satellite observations of the environment's state. All parts of the environment can optionally pipe data to the Vizard visualization engine.

2.3.1 Environment Lifecycle

As shown in Figure 2.3, the lifecycle of the environment consists of instantiation, stepwise execution, and deletion:

(1) **Reset (before):** A new Basilisk simulation is constructed each time the environment is reset. First, the FSW, dynamics, and world models—collections of preexisting Basilisk

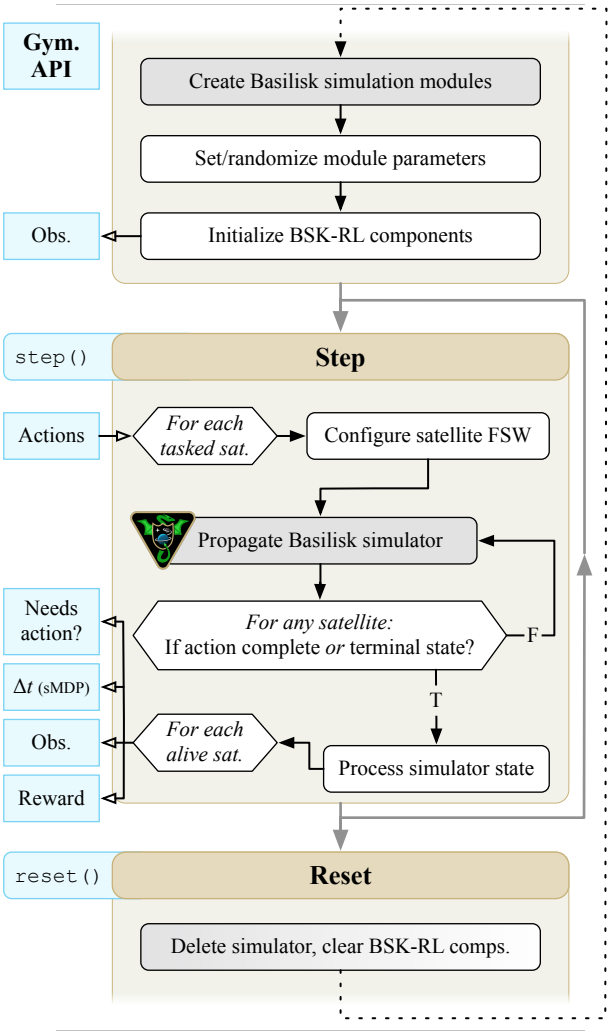


Figure 2.3: Environment lifecycle.

modules—used by each part of the simulator are identified. One is for the "world" that the satellites act in, containing simulations of gravity, the atmosphere, and solar position among other effects. Each satellite has two models associated with it: a dynamics model, which includes physical components and their dynamical interactions, and a FSW model, which consists of algorithms that generate control inputs to the actuators, instruments, and other spacecraft components in the dynamics module.

The models are initialized from a dictionary of simulation parameters that are automatically generated from the selected models. These values include mass properties, power draws, data rates, orbit parameters, epoch, and FSW algorithm settings, among others. A dictionary of default values for the world model and each satellite's FSW and dynamics models can be easily generated and modified. This allows the user to see how the simulation can be configured without *needing* to set all parameters. Parameters can be set directly by value or with functions that randomize the parameter on each reset. The latter is useful for randomizing initial conditions, such as the epoch and orbit, in order to learn a policy that generalizes over domains.

Finally, BSK-RL specific components such as the communication, reward and data system, and scenario are reset for the new episode. Examples include generation of targets in an Earth observation environment and computation of inspection objectives in a relative motion environment.

(2) **Step:** The environment is stepped by giving new actions to some or all alive satellites. Actions are converted into commands that reconfigure the FSW modules in the Basilisk simulation for each satellite: Flight software modes are enabled and disabled, and values in the FSW are changed to correspond to the selected action. Examples include setting the pointing target in the attitude controller or enabling imaging by an instrument once thresholds are met. Once the FSW is configured, the Basilisk simulator is propagated one integration step at a time; after each time step, each satellite is checked for action

completion or terminal states. If either occurs for any satellite, the program returns from Basilisk and processes the results of the simulation for all agents. The mission objective data is communicated, the rewards are computed and yielded, and the observation manager generates an observation for each satellite that is still alive. A signal is also returned for each alive satellite indicating whether its previous action was interrupted by another agent or if the action was successfully completed. Accordingly, learning algorithms can be configured to generate new actions only for satellites that are currently without a task, if desired. If no action is passed for a particular satellite, it will continue its previous task. Satellites that encounter a terminal state during the step are removed from consideration.

In light of the variable-duration environment steps, it is possible to view the resulting environment as a semi-Markov decision process (sMDP), which groups MDP substeps into multi-step actions or tasks. To support this, the Δt of the step in seconds is returned as extra information from the step function.

(3) **Reset (after):** When the environment is reset, the Basilisk modules and simulation manager are deleted to prevent memory leaks and ensure independence across episodes. Basilisk retains certain data in memory such as SPICE kernels, preventing overhead from expensive file operations.

2.3.2 Basilisk Simulation Models

As shown in Figure 2.2, the Basilisk simulation used in the BSK-RL environment is composed of three types of models, each of which consists of configured and connected Basilisk modules. The world model defines global effects in the Basilisk simulation. The dynamics model includes the physical components for each satellite, including actuators, power and data storage subsystems, and rigid body dynamics. The FSW model for each satellite consists of algorithms that control actuators and other subsystems that are present in the dynamics. When certain model types depend on each other (i.e. drag dynamics requiring a world model with an atmosphere or instrument control

FSW requiring an instrument in dynamics), this is documented and programmatically enforced. Within a class of models, different subclasses can be combined through multiple inheritance to produce a model with multiple features.

For each type of model, a table of implemented models is provided; further details on these can be found in the BSK-RL documentation. Predefined models configured with custom parameters allow for a wide variety of environments. The user can also define their own models based on the abstract model type requirements and compose them with existing models. This allows for the use of mission-specific FSW algorithms and satellite architectures.

2.3.2.1 World Models

A single world model exists for each BSK-RL environment containing common effects that act on all satellites, such as gravity and occlusion. World models have limited required machinery, just requiring a setup function to be defined in the abstract class definition. A full list of implemented world models is given in Table 2.3.

Table 2.3: World models implemented in BSK-RL.

Model	Description
`WorldModel`	SPICE-based celestial bodies, spherical harmonic Earth gravity.
`EclipseWorldModel`	Sun-Earth occlusion model for satellite eclipse evaluation.
`AtmosphereWorldModel`	Exponential atmosphere for drag effectors.
`GroundStationWorldModel`	Predefined or custom ground station locations for up/downlink.

2.3.2.2 Dynamics Models

Each satellite has a dynamics model that represents physical components and effects. Physical bodies (bus, solar panels, reaction wheels, etc.) ultimately use Basilisk's BSM for coupled dynamics. The dynamics models expose readable properties that can be used in the observation, such as perfect knowledge of the satellite position, attitude, and subsystem states. A full list of implemented

dynamics models is given in Table 2.4.

Dynamics models continuously check for spacecraft failure states, which indicate that a constraint has been violated. Functions decorated with `@aliveness_checker` are evaluated at each step to ensure that the state of the Basilisk simulation is valid. As a result, arbitrary constraints can be easily defined. Commonly used constraints include keeping a positive battery charge level, avoiding reaction wheel saturation, and maintaining orbital altitude. In the single-agent environment, failure terminates the episode; in the multi-agent environment, failure removes the failed satellite from the list of active agents. Optionally, a penalty can be subtracted from the step reward on failure in order to discourage such behavior.

Table 2.4: Dynamics models implemented in BSK-RL.

Model	Description
`DynamicsModel`	Base class with rigid hub; observation properties for attitude and orbit.
`BasicDynamicsModel`	Satellite with common features including reaction wheels, RCS thrusters, and a power subsystem; also inherits from `Eclipse`, `DisturbanceTorque`, and `AtmosphericDrag` models.
`EclipseDynModel`	Connects the position to the `EclipseWorldModel` for illumination factor computation.
`DisturbanceTorqueDynModel`	Applies a constant disturbance torque to the satellite.
`AtmosphericDragDynModel`	With the atmosphere model, applies facet-based drag force and torque.
`LOSCommDynModel`	Evaluates line-of-sight communication availability between satellites.
`ImagingDynModel`	Agile pointing instrument, data storage system, downlink transmitter.
`ContinuousImagingDynModel`	Nadir scanning instrument, data storage system, downlink transmitter.
`ConjunctionDynModel` and `MaxRangeDynModel`	Enforces keep-out and keep-in regions relative to another satellite.
`RSODynModel` and `RSOInspectorDynModel`	Discretized inspection areas on the RSO spacecraft.

2.3.2.3 FSW Models

Table 2.5: FSW models implemented in BSK-RL.

Model	Description
`FSWModel`	Nadir-pointing and uncontrolled flight modes; low fidelity attitude control model.[a]
`BasicFSWModel`	Sun pointing and reaction wheel desaturation modes; reaction wheel-based feedback control.
`ImagingFSWModel`	Adds agile pointing control reference generation, data downlink mode, and imaging instrument activation based on attitude errors.
`ContinuousImagingFSWModel`	Adds imaging instrument control to the nadir-pointing mode.
`SteeringFSWModel`	Replaces feedback attitude control law with steering law [131].
`MagicOrbitalManeuverFSWModel`	Low-fidelity impulsive orbital maneuver control model.
`RSOInspectorFSWModel`	Adds a tracking reference for pointing at and imaging another spacecraft.

[a] This attitude control model "snaps" the true attitude of the satellite to the reference, rather than dynamically control it with actuators.

Each satellite has a FSW model that defines configurable flight modes—collections of algorithms—that control the behavior of the actuators and subsystems in the dynamics model. Algorithms range from control laws that generate reference states and actuator commands to control of instruments and transmitters. This model represents the guidance, navigation, and control (GNC) and flight operations portions of the FSW, not a full embedded FSW stack. The FSW model exposes an interface between the Basilisk simulation and BSK-RL via functions decorated with `@action`. These functions turn on and off flight modes and configure specific algorithms to make the satellite behave in a particular manner; this is akin to getting a command from the ground or an onboard autonomy manager. As with the dynamics models, readable properties are exposed for use in observations. While no attitude or orbit determination algorithms are currently included in the package, they could be added to the FSW stack alongside simulated measurements in the dynamics model as a more realistic alternative to using perfect-knowledge values from the dynamics in observations. A

full list of implemented FSW models is given in Table 2.5.

2.3.3 Satellite Interface Definition

A satellite is the basic agent unit in the environment. The observations visible to the agent and the actions it can take are defined by the satellite class. Satellite configurations are defined as subclasses to streamline the creation of multiple satellites of the same type in constellation simulations, or to reuse them across experiments. A satellite subclass defines a list of observation and action objects as class properties, as well as specifying the dynamics and FSW models to use in the Basilisk simulation. The following sections examine the satellite class configuration in Listing 2.1 in more detail.

```
class MyScanningSatellite(AccessSatellite):
    observation_spec = [
        obs.SatProperties(
            dict(prop="battery_charge_fraction"),
            dict(prop="storage_level_fraction", norm=0.1),
            dict(
                prop="max_wheel_speed",
                fn=lambda sat: max(sat.dynamics.wheel_speeds_fraction),
            ),
        ),
        obs.OpportunityProperties(
            dict(prop="opportunity_open", norm=T_ORBIT),
            dict(prop="opportunity_close", norm=T_ORBIT),
            type="ground_station",
            n_ahead_observe=1,
        ),
        obs.Eclipse(norm=T_ORBIT),
        obs.Time(),
    ]
    action_spec = [
        act.Scan(duration=180.0),
        act.Charge(duration=120.0),
        act.Downlink(duration=60.0),
        act.Desat(duration=60.0),
    ]
    dyn_model = (ContinuousImagingDynModel, GroundStationDynModel)
    fsw_model = (ContinuousImagingFSWModel)
```

Listing 2.1: Example of satellite class configuration.

The class from Listing 2.1 can then be instantiated to create an agent for the environment. Listing 2.2 demonstrates how a parameter can be set as a constant on configuration (as demonstrated for `mass`) or randomized every episode (as with `dragCoeff`). The latter option allows for power-

ful domain randomization configuration [132]. Advanced utilities for correlated randomization of parameters across satellites are also implemented and described in the documentation.

```
sat_args = dict(mass=300, dragCoeff=lambda: np.random.uniform(2.0, 2.4))
env = SatelliteTasking(
    satellite=MyScanningSatellite(name="Scanner-1", sat_args=sat_args),
    ...
)
```

Listing 2.2: Example of satellite instantiation.

2.3.3.1 Observation Specification

The observation is specified as a list of observation objects. These can be configured to fetch values from the models that make up the Basilisk simulation and other abstractions within the environment, ranging from observations and transformations of physical properties to descriptions of objective availability and completion. Normalization of these values, which is generally necessary in deep reinforcement learning (DRL) [133], is also supported. The observation space is automatically inferred from the specification. The resulting observations can be returned in a flattened array or as a human-readable dictionary; this yields compatibility with RL libraries while allowing for easy debugging and interpretation. Implemented observations are described in Table 2.6, and abstract observation classes are provided to create custom components.

Table 2.6: Observations implemented in BSK-RL.

Observation	Description
`SatProperties`	Extract and normalize arbitrary properties from the satellite's FSW and dynamics models. Common properties are defined in models; arbitrary functions of the satellite can also be specified.
`RelativeProperties`	Observations of the satellite relative to some other chief satellite; like `SatProperties`, supports predefined and arbitrary functions.
`OpportunityProperties`	Information about the next N along-track location access opportunities of a particular type (i.e. imaging target, ground station).
`Eclipse`	Time until the start and end of the soonest (or current) eclipse.
`Time`	Current time in the simulation, by default normalized by episode length.

An example observation specification is given in lines 2 to 19 of the `MyScanningSatellite` definition. In this specification, a number of native dynamics and FSW properties are selected and normalized, as well as an arbitrary function to fetch `max_wheel_speed`. `OpportunityProperties` is configured to return the open and close times of the next ground station availability. The next eclipse times and elapsed time in the episode are also included.

2.3.3.2 Action Specification

Similar to the observation, the actions available to the learning agent are specified via configurable action objects in the satellite definition. In general, actions take the policy output and convert it to a FSW configuration consisting of active modes and parameters that are used when the Basilisk simulation is executed. BSK-RL natively implements discrete and continuous action spaces with a variety of actions of each type. Abstract types for defining new actions of those types are provided as well as abstract frameworks for implementing new action spaces. For example, hybrid discrete-continuous spaces have been used in recent research [134].

Discrete Actions Discrete actions are specified as a list of discrete action objects. Each object can add one or multiple actions to the action space. Many actions enable a certain mode for a set duration, but some have more complex behavior associated with them. Implemented discrete actions are listed in Table 2.7. In lines 20 to 25 of the example configuration, a simple four-mode action space is configured, consisting of nadir scanning, charging, downlinking, and desaturating reaction wheels.

Continuous Actions For lower-level control and path planning tasks, continuous action spaces are also implemented. A single continuous action type (which may have multiple dimensions) can be configured in the satellite class definitions. Table 2.8 lists implemented continuous actions.

2.3.4 Objective Abstractions

The remaining components of the environment define how agents learn about and are rewarded for the abstract mission objectives.

Table 2.7: Discrete actions implemented in BSK-RL.

Action	Dim.	Description
`Drift`	1	All FSW is turned off and the satellite is passive.
`NadirPoint`	1	Point nadir using the control law in the FSW model.
`Charge`	1	Point body-fixed solar panels at the Sun; actual charging is dependent on eclipse and attitude.
`Desat`	1	Fires the reaction control thrusters to desaturate the reaction wheels.
`Downlink`	1	Enable the downlink transmitter and associated power costs; must be in range of a ground station to actually offload data.
`Image`	N	Creates N actions. Points the instrument at the n^{th} target along-track and collects an image once in range and pointing threshold. Completes as soon as the image is taken.
`Scan`	1	Point the instrument nadir and continuously collect data once attitude is within bounds.
`Broadcast`	1	Share scenario data with other satellites, as described for `BroadcastCommunication`.

Table 2.8: Continuous actions implemented in BSK-RL.

Action	Dim.	Description
`ImpulsiveThrust`	4	Instantaneously change the satellite's velocity (3 elements) and then wait for some duration (1 element). A FSW mode to enable while waiting can be specified.
`ImpulsiveThrustHill`	4	`ImpulsiveThrust` with the thrust vector supplied in the Hill frame.
`AttitudeSetpoint`	4	Set the attitude control algorithm's reference and then wait for some duration.

2.3.4.1 Scenario

The scenario represents the mission objectives that are of interest to agents. In Earth observation tasks, the scenario includes locations, priorities, and constraints on imaging targets. In a rendezvous or docking task, the scenario would contain information about which satellites are trying to reach each other and any constraints or requirements for the approach. For resident

space object (RSO) inspection tasks, the scenario would capture what facets of the RSO must be inspected and any constraints on inspection. In general, the scenario object is very flexible and has no methods that must be implemented. Implemented scenarios are listed in Table 2.9.

Table 2.9: Scenarios implemented in BSK-RL.

Scenario	Description
`UniformTargets`	Ground imaging locations uniformly distributed over Earth.
`CityTargets`	Ground imaging locations distributed in clusters near cities.
`UniformNadirScanning`	Surface of Earth with uniformly important features to scan.
`SphericalRSO`	Discretized inspection points on a spherical RSO.

2.3.4.2 Data & Rewarder

The data and rewarder system translates from the underlying Basilisk simulation to scalar rewards. Three classes are implemented: `Data`, a unit of data or progress; `DataStore`, a per-satellite component that generates data from the simulator state; and `GlobalReward`, a component that distributes rewards to each agent based on newly created data:

- `Data` represents a unit of progress towards the task, and must implement an addition function to combine it with other data. In an Earth observation or RSO inspection task, the data may include what has been imaged and information about those images. In other cases, the data may be more abstract: for example, rendezvous task data may just be the current relative state between the agent and the target.

- The `DataStore` generates and stores data on a per-satellite basis. The data store must implement a function that compares the environment at the previous and current step and produces the new data that has been created. Information that a satellite is aware of in the data store may impact its actions: for example, a satellite would only know to exclude an already-satisfied task from its possible task list if the task's completion is represented in the data store.

- The `GlobalReward` acts akin to a global critic for the environment. Given the per-agent new data at a given step and the current state of the environment, rewards are calculated for each agent. While satellites should only act based on their local state, the rewarder computes reward based on the global state. For example, in an agile Earth-observing satellite (AEOS) scenario, the rewarder yields reward based on priority for each previously unfulfilled request fulfilled, regardless of whether a particular satellite knows that status. Instances of `GlobalReward` are passed to the environment constructor explicitly, and the data store and data are inferred from the selected `GlobalReward`.

Implemented reward systems are listed in Table 2.10, and new reward systems can be defined by inheriting from the base classes.

Table 2.10: Reward and associated data systems implemented in BSK-RL.

Reward	Description
`UniqueImageReward`	Rewards as a function of priority for the first image of each ground target.
`ScanningTimeReward`	Rewards proportional to amount of nadir scanning completed.
`ResourceReward`	Rewards based on an arbitrary function of satellite state; implement penalties and bonuses for using or gaining a resource such as fuel, battery, distance, or time.
`RSOInspectionReward`	Rewards for inspection of new portions of an RSO, with a bonus for complete inspection.

2.3.4.3 Communication

In some scenarios, knowing what data has been collected by other agents is important for completing a goal. Transmitting data may prevent duplication of effort on already-complete tasks or inform other agents about tasks that require collaboration to satisfy. Toward this, a general notion of communication is implemented for multi-agent scenarios. Each communication method generates a directed graph of which pairs of agents should share information at the end of each step. Implemented methods are listed in Table 2.11. As with other components, custom methods

can be developed by inheriting from the base class; custom communication requires a function that returns which satellites share data based on the current environment state.

Table 2.11: Communication types implemented in BSK-RL.

Communication	Description
`NoCommunication`	No data sharing.
`FreeCommunication`	Every agent shares with every other agent.
`LOSCommunication`	Shares bidirectionally between satellites with `LOSCommDynModel` that have a direct line-of-sight.
`BroadcastCommunication`	Shares from satellites that used the `Broadcast` action during the prior step.

2.3.5 Visualization

All components of a BSK-RL simulation can pass data to Basilisk's companion visualization tool Vizard,[4] allowing the user to view results as shown in Figure 2.1. In order to prevent additional overhead in training, Vizard-related components are completely disabled unless the environment is explicitly configured to output a Vizard file. Vizard outputs are useful for validating, debugging, and understanding the performance of environments configured in BSK-RL.

[4] https://avslab.github.io/basilisk/Vizard/Vizard.html

Chapter 3

The Agile Earth Observation Satellite Scheduling Problem

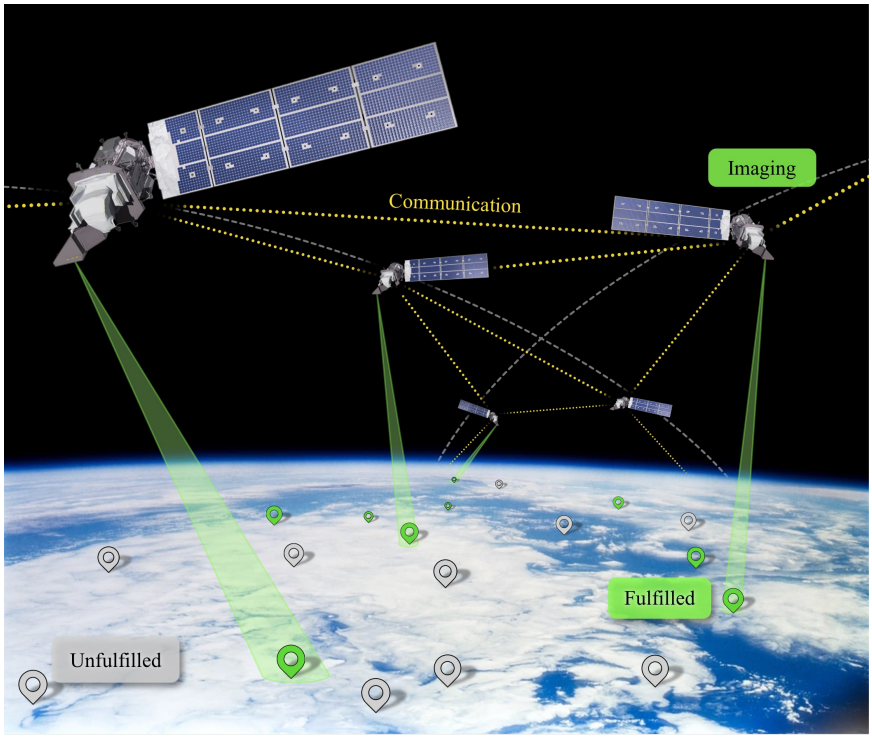


Figure 3.1: Concept art of the agile Earth-observing satellite scheduling problem (AEOSSP).

3.1 Motivation

On-demand satellite imagery is beneficial for scientific observations, commercial operations, disaster response, reconnaissance missions, and many other purposes [2]. Agile satellites are well-suited to these tasks as they can collect many images across large regions: They are capable of

performing along-track slews to image requests in front and behind them, as opposed to traditional Earth-observing satellites (EOSs) that can only slew across-track, allowing for more images to be collected.

The objective of the AEOSSP, illustrated in Figure 3.1, is to sequence operations such that the value of images collected by an agile Earth-observing satellite (AEOS) is maximized. While agility can greatly increase the imaging throughput of a satellite, it complicates the planning and scheduling problem because both the order in which requests are fulfilled and the time of fulfillment within the access window must be specified. As a result, the solution space for possible schedules is large and complex, containing discrete and continuous variables [6].

This chapter develops the AEOSSP for one or more satellites—sometimes known as the agile Earth-observing constellation scheduling problem (AEOCSP)—and introduces variations studied in this work.

3.2 Problem Statement

In this section, a basic request-sequencing-only version of the AEOSSP is described for N observing satellites and *a priori* knowledge of fixed request priorities and locations. Resource constraints are not yet considered.

3.2.1 Request Model

Image requests are specified using a point-based request model as a tuple ρ_i consisting of an Earth-fixed location $\boldsymbol{r}_i$ and a priority r_i in the set of all requests $\mathcal{R}$:

$$\rho_i = (\boldsymbol{r}_i, r_i) \in \mathcal{R} \tag{3.1}$$

Higher priorities are more desirable. All requests start in the global unfulfilled set $\mathcal{U}$. If the location is successfully imaged, the corresponding request is removed from the unfulfilled set $\mathcal{U}$ and added to the set of fulfilled requests $\mathcal{F}$.

$\mathcal{U}$ and $\mathcal{F}$ reflect the true global state of the request lists. When considering closed-loop

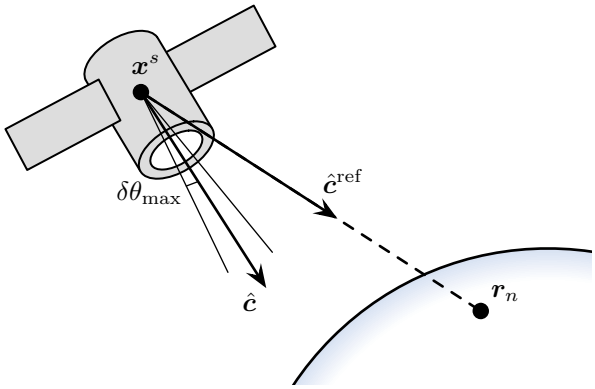


Figure 3.2: AEOS geometry when fulfilling a request.

tasking, each satellite s maintains its own sets $\mathcal{U}_s$ and $\mathcal{F}_s$. Depending on the availability of communication in the environment, each satellite's list is updated as new information about the other satellites' actions becomes available.

3.2.2 Satellite Dynamics

Each satellite travels in a fixed orbit around the Earth. For each request, each satellite s has a precomputable set of opportunity windows $(t^o_{s,i}, t^c_{s,i}) \in \mathcal{O}_{s,i}$ for each request ρ_i. In this work, only a minimum elevation angle constraint $\phi_{\min}$ between the target and satellite must be satisfied, meaning that imaging windows are determined by a view cone about the satellite nadir. However, all the methods presented to solve the problem admit the use of any constraints that are reducible to restrictions on imaging time and can be computed *a priori*.

To fulfill requests, the satellite with position $\boldsymbol{x}_s(t)$ uses an imaging instrument pointing in the body-fixed $\hat{\boldsymbol{c}}$ direction. When fulfilling a request, the satellite must point the instrument in the direction of the target

$$\hat{\boldsymbol{c}}^{\text{ref}}(t; s, i) = \frac{\boldsymbol{r}_i - \boldsymbol{x}_s(t)}{|\boldsymbol{r}_i - \boldsymbol{x}_s(t)|} \tag{3.2}$$

within a pointing angle error $\delta\theta_{\max}$,

$$\angle(\hat{\boldsymbol{c}}, \hat{\boldsymbol{c}}^{\text{ref}}) < \delta\theta_{\max} \tag{3.3}$$

The instrument must also track the target with a reference body rate $\boldsymbol{\omega}_{\mathcal{B}/\mathcal{P}}$ between the body frame

$\mathcal{B}$ and planet frame $\mathcal{P}$ of

$$\boldsymbol{\omega}^{\text{ref}}_{\mathcal{B}/\mathcal{P}}(t;s,i) = \frac{\frac{\mathcal{P}_{\text{d}}}{\text{d}t}\boldsymbol{x}_s(t) \times \hat{\boldsymbol{c}}^{\text{ref}}(t,i)}{|\boldsymbol{r}_i - \boldsymbol{x}(t)|} \tag{3.4}$$

where $\frac{\mathcal{P}_{\text{d}}}{\text{d}t}$ is a $\mathcal{P}$-frame derivative, within a rate error $\delta\omega_{\max}$,

$$|\boldsymbol{\omega}_{\mathcal{B}/\mathcal{P}} - \boldsymbol{\omega}^{\text{ref}}_{\mathcal{B}/\mathcal{P}}| < \delta\omega_{\max} \tag{3.5}$$

In order to fulfill a request, the angle and rate error thresholds must be met at a time t during an open opportunity for the satellite-request combination, satisfying

$$\exists(t^o_{s,i}, t^c_{s,i}) \in \mathcal{O}_{s,i} \text{ such that } t^o_{s,i} \leq t \leq t^c_{s,i} \tag{3.6}$$

When planning to fulfill a request ρ_i at some point in the future $t_{s,i}$, a satellite s must have sufficient time to slew from its current attitude $sigma_0$ to one satisfying the tracking errors at the desired fulfillment time. The slew transition time, the function Δt_{slew}, must be less than the duration between the current time t_0 and the planned request visitation time t_i:

$$\Delta t_{\text{slew}}(\boldsymbol{\sigma}_0, \boldsymbol{\omega}_{\mathcal{B}/\mathcal{P},0}, \boldsymbol{x}_s(t), \boldsymbol{r}_i) \leq t_i - t_0 \tag{3.7}$$

The function Δt_{slew} is challenging—if not impossible—to find analytically due to the time-varying tracking reference and is highly dependent on the attitude control law implemented on the satellite. By imposing a few reasonable assumptions, the function gains properties that are useful when applied to the planning task:

(1) The attitude controller is deterministic.

(2) The controller performs identically regardless of rotation about the boresight axis (e.g. an inertia-compensated controller that slews in an axis-agnostic manner) *or* the controller maintains a fixed about-boresight rotation relative to the trajectory (e.g. a controller that maintains zero yaw in the Hill frame).

(3) As previously stated, the satellite's position trajectory is a known fixed orbit.

(4) Once the satellite has settled its instrument on a target, it can indefinitely track the target (ignoring visibility constraints)

With these assumptions, Equation 3.7 reduces to the simpler form

$$\Delta t_{\text{slew}}(\hat{\boldsymbol{c}}_0, \boldsymbol{\omega}_{\mathcal{B}/\mathcal{P},0}, t_0, \boldsymbol{r}_i) \leq t_i - t_0 \tag{3.8}$$

which conveniently consists of inputs that can be computed from attitude references for a given request (Equation 3.2 and Equation 3.4).

In the satellite configuration used in simulation, these assumptions are satisfied. Four reaction wheels are used for attitude control, producing a relatively high control torque $u_{\max}$ to meet the agility requirements of the mission. An exponentially stable controller, as defined in Reference 131, that generates and tracks an attitude trajectory in modified Rodrigues parameter (MRP) space is executed by the satellites in this chapter. Additionally, the control torque is clipped by the maximum torque of the reaction wheels when performing maneuvers in the simulation. Even with this combination of factors, the nonlinear controller satisfies the two requirements of the method: maneuvers are deterministic and kinematically identical for any about-boresight rotation; thus, the slew times can be expressed by Equation 3.8.

3.2.3 Optimization Objective

The goal of the AEOSSP is to maximize the sum of priorities of uniquely fulfilled requests. The value of fulfilling $\rho_i \in \mathcal{U}$ yields a reward equal to the priority r_i; fulfilling an already-fulfilled request $\rho_i \in \mathcal{F}$ yields no reward. Should an operator want to image the same location multiple times, multiple requests can be made in that location with constraints on the availability period of each request.

Mathematically, the objective of the problem is to maximize the sum of the fulfilled priorities subject to the transition dynamics of the constellation environment. The optimization is performed

over the task sequence S^s, an ordered set of requests, for each satellite:

$$\underset{\{S^1,\ldots,S^N\}}{\text{maximize}} \quad \sum_{S^s} \sum_{\rho_i \in S^s} r_i \tag{3.9}$$

$$\text{such that} \quad S^s \text{ feasible w.r.t. } \Delta t^{\text{slew}} \qquad \forall s \in 1, \ldots, N \tag{3.10}$$

$$\rho_i \notin S^{s'} \qquad \forall\, S^{s'} \neq S^s \quad \forall \rho_i \in S^s \quad \forall s \in 1, \ldots, N \tag{3.11}$$

Beyond the numerical objective (Equation 3.9), feasibility requirements (Equation 3.10), and nonduplication constraint (Equation 3.11), other considerations exist for "good" solutions: avoiding unsuccessful attempts (i.e. scheduled tasks with insufficient time to slew) and achieving an equitable division of tasks in multi-agent scenarios.

3.3 Simulation Parameters

The scenario described in the problem formulation is modeled using BSK-RL. Of particular relevance to this work, attitude dynamics are modeled to a high fidelity: rigid-body dynamics models of the spacecraft bus and four reaction wheels are computed using the back-substitution method [121]. The wheels are driven by flight-proven control software that feeds torque commands to the reaction wheels [131]. The environment uses a SPICE-based model of planetary motion and gravitation for orbital dynamics propagation. Satellite flight software (FSW) and dynamics are integrated at 2 Hz.

3.3.1 Satellite Configuration

The satellite's orbit is circular with a fixed inclination and altitude and randomized true anomaly and ascending node. At most one day of planning (15 orbits) is considered, as is typically considered to be the practical limit for non-adaptive preplanning. Important satellite parameters are given in Table 3.1. Other satellite parameters are the defaults used in BSK-RL; the specific Basilisk modules used by the satellite can also be found in the repository.

In multi-satellite AEOSSP scenarios, two parameterized constellation architectures are considered:

Table 3.1: Satellite parameters for the AEOSSP.

Parameter	Value(s)
Horizon H	≤ 1 day (15 orbits)
Inclination	45°
Altitude	800 km
$\delta\theta_{\max}$	0.01 MRP norm (2.29°)
$\delta\omega_{\max}$	0.01 rad/s (0.57°/s)
$u_{\max}$	0.4 N·m (per-axis)
Mass Properties $m, \boldsymbol{I}$	330 kg, $[82.1, 98.4, 121.0]$ kg·m^2
Gains K_1, K_3 [131]	0.25, 3.0
$\omega_{\max}$ [131]	5 rad/s

- **String:** A lead-follower "string-of-pearls" type constellation with N satellites in a 800km $\times$ 45° orbit separated by Δf true anomaly.
- **Walker:** A Walker-Delta constellation [135] with N satellites evenly distributed across four equally spaced planes in an 800km $\times$ 45° orbit. An example of a Walker-Delta constellation is shown in Figure 7.13.

3.3.2 Request Configuration

Representative sets of requests are generated for experiments. Two distributions of request locations across various total request counts are considered, as shown in Figure 3.3: a uniform distribution around the globe and a clustered distribution in which city locations[1] serve as a proxy for image request frequency, inspired by Eddy and Kochenderfer [136]. The city-based distribution is intended to be representative of many Earth observation missions, where the request type tends to be concentrated in specific areas of interest (e.g. coastline, urban areas, etc.). In the latter case, the satellite can travel for half an orbit without encountering any request, then have hundreds available within a few minutes over very populous areas. A lower maximum request count is considered for

[1] City location data from simplemaps.com, Basic World Cities Database v1.901, CC BY 4.0.

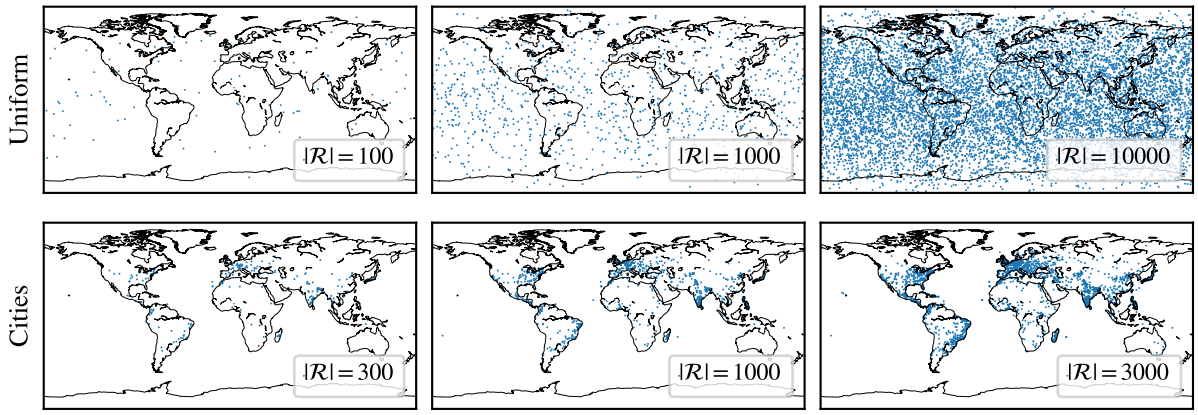


Figure 3.3: Examples of request distributions at different densities.

cities to keep the maximum local request density similar. Higher local request densities produce impractical combinations that could be combined into single requests, given the instrument's field of view (FOV). In both cases, each initialization of the environment at a given density randomizes the locations of uniform points or the subset of cities selected.

For this study, imaging requests are created with relatively simple visibility constraints. Only a minimum-elevation-angle constraint ϕ_{min} between the target and satellite must be satisfied, meaning that imaging windows are determined by a view cone about the satellite nadir. Since request limitations only impact the preprocessing step of window generation, more exotic *a priori* constraints could be applied without limiting the applicability of the methods considered in this work. Request rewards are randomized over a uniform distribution $r \in [0, 1]$. A complete listing of request parameters is given in Table 3.2.

3.4 Variations of the AEOSSP

Two variations of the AEOSSP are introduced and considered: one with time-dependent request priorities, and another with power constraints.

Table 3.2: Request parameters for the AEOSSP.

Parameter	Value(s)	Description
$\phi_{\min}$	58°	minimum elevation angle
r_{FOR}	500 km	field of regard (FOR) radius (from $\phi_{\min}$)
r	Uniform$(0, 1)$	request value
Distribution	$\{\text{uniform}, \text{cities}\}$	see Figure 3.3
$\lvert\mathcal{R}\rvert$	$[100, 10000]$	number of requests (uniform)
$\lvert\mathcal{R}\rvert$	$[300, 3000]$	number of requests (cities)

3.4.1 Time-Dependent Priorities

A more general form of requests introduces time-dependent priorities, where the value of a request depends on both when it is fulfilled and which satellite fulfills it. Instead of ρ_i having fixed priority r_i, it has priority $r_i(s, t)$. As a result, the optimization objective (Equation 3.9) is performed for $S^s = [(\rho_i, t_i^s), ...]$, ordered sets of requests and times:

$$\underset{\{S^1,...,S^N\}}{\text{maximize}} \sum_{\rho_i \in \mathcal{F}} r_i(s, t_i^s) \tag{3.12}$$

This time- and satellite-dependent formulation allows for the encoding of many arbitrary image-quality-like costs and constraints. Since the trajectories of the satellites are fixed, elevation angle ϕ is a function of time and therefore squint-angle-dependent rewards reduce to time-dependent rewards. Likewise, properties that vary more slowly than the interval of one opportunity window can also impact request priority in a time-varying way, such as illumination angle requirements.

When time-dependent rewards are considered in this work, the value is penalized as a function of elevation angle ϕ:

$$r_n(\phi) = \frac{r_n \phi}{\pi/2} \tag{3.13}$$

As with the opportunity window requirements, only elevation-angle-based time-dependent priorities are considered in this work, but all the methods presented generalize to arbitrary functions of time for opportunities and rewards.

Table 3.3: Parameters for the power subsystem of the power-constrained AEOSSP.

	Parameter	Value(s)
Power	$z_{\max}$	120 W·h
	z_0	$[0.4, 1.0] \times z_{\max}$
	$\dot{z}_{\text{base}}$	20 W
	$\dot{z}_{\text{im}}$	10 W
	η_{rw}	0.5

3.4.2 Power-Constrained Scheduling

The AEOSSP is commonly subject to resource constraints that need to be maintained by the scheduler in addition to the science objective. Representative of this, a variant of the AEOSSP is proposed in which the satellite has a power subsystem that must be managed, adding the constraint on battery level z

$$\text{such that} \quad z_s(t) > 0 \qquad \forall s, t \tag{3.14}$$

to the optimization problem in Equation 3.9.

To model the power system, each satellite is equipped with a battery with capacity $z_{\max}$ and solar panels with efficiency η and area A with their normal antiparallel to the instrument pointing direction. The satellite has a baseline power draw $\dot{z}_{\text{base}}$, a reaction wheel power draw $\dot{z}_{\text{rw}}(\mathbf{\Omega})$, and, when the imaging instrument is turned on, an additional draw $\dot{z}_{\text{im}}$. The power generation of the solar panels is a function of the attitude over time and the eclipse state. As a result, the power state over time is highly dependent on the actions taken.

The power-constrained environment uses the same simulation parameters as the sequencing environment (Table 3.1) with the addition of the power subsystem parameters in Table 3.3. Power generation using the solar panels is modeled based on solar incidence angle due to the spacecraft attitude and eclipse state. The power draw of the satellite is a function of the reaction wheel speeds (with η_{rw} representing the electrical-to-mechanical power conversion efficiency) and, when

imaging, the instrument power draw. As with the power-free environment, the power-constrained environment can be found in the BSK-RL repository.

Chapter 4

Mixed-Integer Scheduling for the AEOSSP

4.1 Motivation

While this dissertation is primarily focused on learning-based methods, it is necessary to first establish a baseline for comparison. For the agile Earth-observing satellite scheduling problem (AEOSSP), this is achieved via mixed-integer programming. While this is an established approach for the problem, existing literature lacks scalability to large problem sizes and accurate representations of the transition dynamics. This chapter develops formulations and algorithms that improve the accuracy and scalability of these methods.

Graph-based representations of the AEOSSP, in which imaging actions are represented by vertices and feasible slews between requests are represented by edges, are common in the literature. Gabrel [37] offers an early graph-based treatment of the problem that assumes a simple transition time model and is computationally limited to small problem instances; additionally, unlike more recent work, off-the-shelf optimization frameworks are not used. Augenstein [39] leverages the properties of directed acyclic graphs (DAGs) representing this problem. Augenstein's formulation treats each imaging event as a single vertex (as opposed to multiple vertices, each representing a different imaging time within an opportunity window). As a result, Augenstein is able to use a simple dynamic programming algorithm for optimization. Eddy's [136] exploration of efficient planning for constellations takes a similar approach to single-point request representation, focusing more on the deconfliction of requests between satellites than individual agile imaging optimization. Peng [137] briefly considers a graph-based method that uses multiple vertices per observation window,

as in this work; this introduces challenges that prevent the use of simpler dynamic programming solutions for path maximization such as those used by Augenstein, and necessitates the use of other combinatorial optimization methods.

Iterative local search and genetic algorithms are good at quickly finding near-optimal solutions to highly combinatorial problems such as the AEOSSP, though they lack the optimality guarantees of mixed-integer programs (MIPs). Lemaître [38] offers one of the first local search approaches to an alternate formulation of the problem that considers strip imaging instead of point imaging, in comparison with other methods. Dilkina [138] compares a variety of early approaches including iterative local search (ILS) and genetic methods to the problem for a single satellite over small (6 hour, $\sim$1000 request) test cases. Mao [139] gives another genetic algorithm-based method for multi-objective optimization, but does not report on specific scenario performance. More recently, Verbeeck's [140] advances in local search for time-dependent orienteering problems (a class which includes the AEOSSP) have led directly to the development of performant ILS-based AEOSSP solvers by Liu [141] and Peng [137]. These two papers consider both time-dependent transition times and time-dependent rewards, planning in up to medium-sized environments with hundreds of requests over a day-long period in tens to hundreds of seconds.

MIP and mixed-integer linear program (MILP) solutions to the AEOSSP offer an alternative method that quantifies solution suboptimality and—with enough time—finds and certifies optimality. As such, they can both be used practically as a planning algorithm or for benchmarking the performance of other methods relative to optimality. Peng [137] formulates the single satellite problem as a MIP to have an optimal comparison for their local search algorithm that considers time-dependent transitions and rewards, but finds that the formulation is unable to find a solution in a reasonable amount of time or without running out of memory for anything but the smallest instances. Their MILP formulation is comparable to the MILP-D formulation described in this chapter. Cho [41], Chen [142], Kim [42], and Wang [50] use MILP formulations for constellation-wide image and downlink scheduling under various constraints, each using heuristics to initialize the solver for improved speed as multi-satellite problems can quickly become intractably large.

Cho includes continuous models for power and data, and models up to 12 satellites for up to 2 days planning with up to 700 tasks; reasonable, but suboptimal, solutions are found within 10,000 seconds but are not evaluated in a secondary simulation environment. Chen introduces continuous variables to more effectively schedule overlapping requests, considering up to 1000. Kim only considers up to 100 tasks, though includes complex task specifications like stereoscopic imaging. Wang introduces cloud uncertainty and optimizes over reward and riskiness for up to 300 requests. In general, the literature tends to consider at most a few hundred to a thousand requests over a few orbits to a day. In Reference 143, D-SHIELD uses dynamic programming with heuristics to perform distributed scheduling for a constellation with reactive capabilities.

The treatment of transition times in existing literature tends to be simplified compared to the realities of spacecraft dynamics when transitioning from tracking one request to another. The most accurate—but most expensive—approach is to execute a dynamics model to test each transition, as proposed by [39]. Another approach is to use a low-order linear model that considers only the difference in roll angle [41, 42] or the total angle change [141, 137] or other unspecified abstract transition constraints [42, 142]; while these methods are convenient for planning problems, they do not fully capture the nonlinear nature of attitude transitions. An ablation study in this chapter demonstrates how lower order models fall short. Nag [40] uses a discretized model of transitions between various attitudes; this captures more of the dimensions that impact transition dynamics but is still a lower-order and lower-dimensional model.

In this chapter, novel developments are made toward accurately representing and efficiently solving the request sequencing (i.e. not considering resource management) portion of the AEOSSP. First, time-dependent transition times between requests are modeled using a neural network (NN) function approximator, a significant improvement in accurately expressing dynamics over the low-order models for transition times. This improvement in transition time estimation translates to better-performing plans than when using common lower-order models because neither is time wasted due to an overconservative estimator nor are requests missed due to an overly aggressive estimator. The structure of the problem is represented as a sparse graph that accounts for different imaging

times for each request, and an efficient MILP formulation is developed for the sparsified graph that maintains optimality guarantees. Compared to the standard dense graph formulation, the sparse formulation is faster, more memory-efficient, and more scalable to larger problems, remaining tractable for thousands of requests. Formulations for time-independent and time-dependent rewards are developed, allowing for the best performance on a specific problem type. Solutions to the planning problem are evaluated in a full-fidelity spacecraft simulator, demonstrating end-to-end performance of the proposed method as opposed to only analyzing the solution quality in the abstract planning space.

4.2 Problem Statement

The AEOSSP formulation from chapter 3 is solved using this method. To prepare for the mixed-integer formulation, additional notation is introduced.

The AEOSSP optimization problem is rewritten as an optimization over sequences of request visitations. A request visitation $v_i^s = (t_i^s, \rho_i^s)$ for satellite s is a tuple of a visitation time and a request.[1] The objective of the agile multi-satellite request sequencing problem is to determine a feasible sequence of request visitations $S^s = v_1^s, v_2^s, ...$ for each satellite that maximizes the sum of imaging values of the requests satisfied:

$$\underset{\{S^1,\ldots,S^N\}}{\text{maximize}} \quad \sum_{s=1}^{N} \sum_{i=1}^{|S^s|} r_i^s(s, t_i^s) \tag{4.1}$$

$$\text{subject to} \quad \Delta t_{\text{slew}}(v_i^s, v_{i+1}^s) \leq t_{i+1}^s - t_i^s \quad \forall \quad v_i^s, v_{i+1}^s \tag{4.2}$$

$$\rho_i^n \neq \rho_j^m \quad \forall \quad i \neq j \vee n \neq m \tag{4.3}$$

That is, find the sequences that maximize the sum of values at imaging time for each fulfilled request (Equation 4.1) while ensuring that there is sufficient time for each satellite to transition between requests (Equation 4.2) and that each request is fulfilled at most once (Equation 4.3). These are equivalent to Equation 3.9 through 3.11.

[1] Note that ρ_i^s, the i^{th} request visited by satellite s, is not the same as ρ_i, the i^{th} globally-indexed request.

In order to execute v_j, the satellite must be able to transition from its state at t_i to a state that satisfies the collect requirements for ρ_j at t_j. The slew transition time must be less than the time until the planned request visitation, as required by Equation 3.8. This requirement can be written directly as a function of sequential visitation because there is a mapping between two sequential visitations and the arguments to the slew transition time function

$$\Delta t_{\text{slew}}(v_i^s, v_j^s) = \Delta t_{\text{slew}}(\hat{\boldsymbol{c}}^{\text{ref}}(t_i^s; s, i), \boldsymbol{\omega}_{\mathcal{B}/\mathcal{P}}^{\text{ref}}(t_i^s; s, i), t_i^s, \boldsymbol{r}_j^s) \tag{4.4}$$

by substituting the reference attitude and rates (Equation 3.2 and 3.4) into Equation 3.8. This compact form is seen in Equation 4.2.

4.3 Slew Estimation

The introduction of Equation 4.4 motivates the methods employed in the mixed-integer formulation: transitions between many v_i and v_j must be checked for feasibility in order to generate a search space of feasible sequences in which the optimal sequence can be found. However, even with careful choices for which v_i and v_j to check, many evaluations of Δt_{slew} are required. While this is trivial for a toy problem that assumes constant or linear transition times, transitions for flight-like systems in general lack an analytical solution for Δt_{slew}. As shown in the results, approximating transitions with a simplified model leads to a loss of optimality on the true system: underestimates of Δt_{slew} may incorrectly classify infeasible transitions as feasible, while overestimates of Δt_{slew} may lead to idle time.

Running a dynamics simulation of every tested transition is prohibitively expensive. As an alternative with lower computational cost, a NN is trained to estimate Δt_{slew}. This approach allows for a computationally low-cost evaluation of Δt_{slew} for many slews.

4.3.1 Network Architecture

The slew estimation network parameterized by $\boldsymbol{\theta}$ is of the form

$$\Delta \hat{t}_{\text{slew}} \approx \Delta t_{\text{slew}}(\hat{\boldsymbol{c}}_0, \boldsymbol{\omega}_{\mathcal{B}/\mathcal{P},0}, \boldsymbol{x}^s(t_0), \frac{^{\mathcal{P}}\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}^s(t_0), \boldsymbol{r}_n; \boldsymbol{\theta}) \tag{4.5}$$

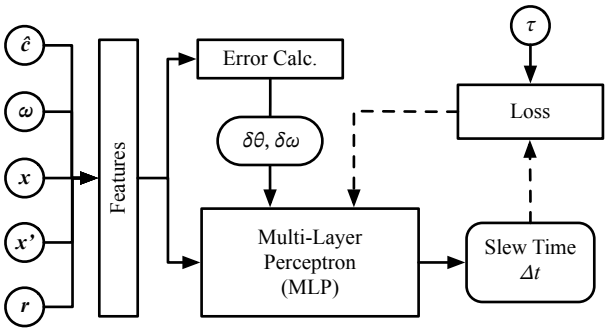


Figure 4.1: Slew transition time estimation network architecture.

with all quantities expressed in the planet-fixed frame $\mathcal{P}$. The initial position and velocity of the satellite are given explicitly, allowing the estimator to generalize to any satellite with the same attitude control and orbital dynamics.

The estimator is a fully-connected feedforward multilayer perceptron (MLP) shown in Figure 4.1. To improve the performance of training, domain knowledge is leveraged: the attitude and rate errors (i.e. the left-hand side of the inequalities in Equation 3.3 and 3.5) are computed from and concatenated with the inputs passed to the MLP. This information is useful for the network since Δt_{slew} tends to be highly correlated with the attitude and rate errors. The other inputs to the network are still necessary to resolve more subtle nonlinearities such as those due to planetary rotation or controller saturation.

4.3.2 Training Procedure

To generate training data, slews to random requests in the upcoming half-orbit along track are executed in the simulation environment, ignoring opportunity window limitations. Satellite states that are inputs to the NN are recorded at various points along each slew, including the initial and final states. The remaining duration of the slew from each point is calculated at the end of each slew for use in regression.

A feedforward MLP with a width of 20 nodes and depth of 8 layers is trained on the state-

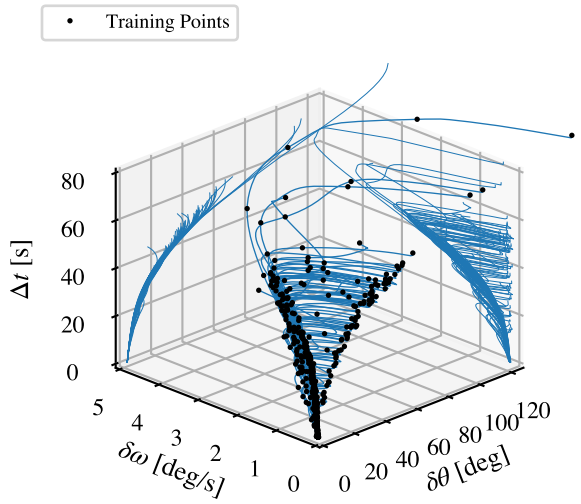


Figure 4.2: A subset of the slews used in training, shown in a lower-dimensional space.

time remaining tuples using the Adam optimizer [35]. An asymmetric loss function [144] combining the properties of root mean squared error and "pinball" losses is used:

$$L(t, \hat{t}; \tau) = \begin{cases} (1-\tau)(\hat{t}-t)^2 & \text{if } \hat{t} \geq t \\ \tau(\hat{t}-t)^2 & \text{if } \hat{t} < t \end{cases} \tag{4.6}$$

The skewness parameter $\tau \in [0, 1]$ can be adjusted to bias the estimator toward overestimation of Δt_{slew} by setting $\tau > 0.5$, which is desirable for this application. An ablation study over τ is presented in subsection 4.3.3.

4.3.3 Slew Estimator Performance

A dataset consisting of state information at 10-second intervals along 1.8×10^5 slews to randomly selected nearby targets is generated, with a subset shown in a lower-dimensional space in Figure 4.2. The network is trained on this dataset with varying loss skewness τ. Figure 4.3 shows the percentage of predictions for the validation dataset that fall within a certain error threshold as a function of τ. As expected, higher values of τ lead to a tendency to overestimate slew durations.

To evaluate the proper tuning of τ, a challenging (i.e. high request density, $|\mathcal{R}| = 10000$)

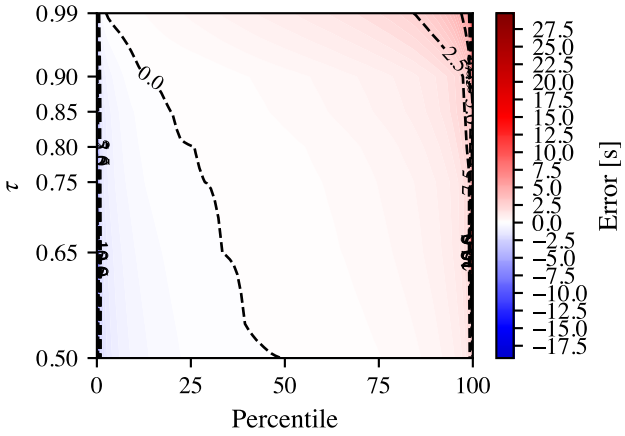


Figure 4.3: Slew estimator prediction errors across 198k validation points as a function of loss skewness τ by percentile of points within error threshold.

simulation environment is instantiated and a "greedy" policy is implemented. In this policy, the satellite is always tasked to image the soonest request that is predicted to be accessible by the estimator. This is a worst-case stress test for the estimator, as the MILP solutions will often favor higher-value requests over earlier ones. The number of successful slews and the total number of requests attempted are recorded over 800 orbits per τ.

Figure 4.4 shows the expected behavior: as the skewness parameter τ increases, the predictor becomes more conservative, attempting fewer requests but succeeding at a higher rate. The value $\tau = 0.75$ is selected for the remainder of the experiments, as it does not show a drop in requests satisfied compared to 0.5 while having a slew success rate $> 99\%$. The estimator takes a few milliseconds to evaluate per batch of 2048 slews; a comparable number of slews would take a few minutes to simulate in Basilisk.

4.4 Optimal Agile Satellite Scheduling

A two-step approach is taken to solve the AEOSSP. First, a graph is constructed for each satellite that represents the feasible transitions between requests. The solution space satisfying Equation 4.2 is abstracted as an edge-weighted DAG. Vertices $v \in V^s$ represent time-request visitation tuples. Visitations with a feasible transition between them are connected by edges $e^s_{i,j} =$

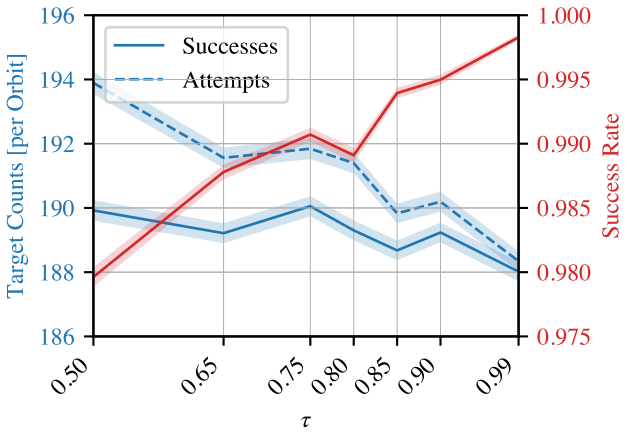


Figure 4.4: Mean slew success rates of a greedy policy using the prediction network with different skewness parameters τ; trials over the same initialization per τ with $|\mathcal{R}| = 10000$; 95% confidence intervals are shown.

$(v_i^s \to v_j^s) \in E^s$ where each edge has a scalar weight $w_{i,j}^s$ representing the value of fulfilling the request at v_j if it has not yet been fulfilled.

Then, a MILP is formulated to find the value-optimal sequence of requests (Equation 4.1 and 4.3) inspired by formulations of the traveling salesman problem (TSP) [45]. For each edge between vertices v_i^s and v_j^s, a binary variable $x_{i,j}^s$ represents the inclusion or exclusion of the transition in the sequence S^s. Vertices are partitioned by request

$$p_n = \{v_i^s \mid \rho_i^s = \rho_n\} \tag{4.7}$$

which allows for the formulation of non-repetition constraints.

Three variations of the process are presented. A naïve approach, MILP-D(ense), for generating and solving a dense graph of feasible slews using the NN-based Δt_{slew} estimator is introduced. A modification to reduce the size of the graph, MILP-S(parse), by removing redundant edges is presented. Finally, a method to account for time-dependent values, MILP-T(ime)D(ependent), is introduced. These methods are summarized in Table 4.1.

Table 4.1: Comparison of MILP formulation complexities.

	Graph $\|V\|$	Graph $\|E\|$	Binary Variables	Time-Dep. Reward
MILP-D	$\mathcal{O}(NH\|\mathcal{R}\|/\delta t)$	$\mathcal{O}(H\|\mathcal{R}\|\|V\|)$	$\|E\| \in \mathcal{O}(NH^2\|\mathcal{R}\|^2/\delta t)$	
MILP-S	$\mathcal{O}(NH\|\mathcal{R}\|/\delta t)$	$\mathcal{O}(\|\mathcal{R}\|\|V\|)$	$\|\mathcal{R}\| + \|E\| \in \mathcal{O}(NH\|\mathcal{R}\|^2/\delta t)$	
MILP-TD	$\mathcal{O}(NH\|\mathcal{R}\|/\delta t)$	$\mathcal{O}(\|\mathcal{R}\|\|V\|/\delta t)$	$2\|E\| \in \mathcal{O}(NH\|\mathcal{R}\|^2/\delta t^2)$	✓

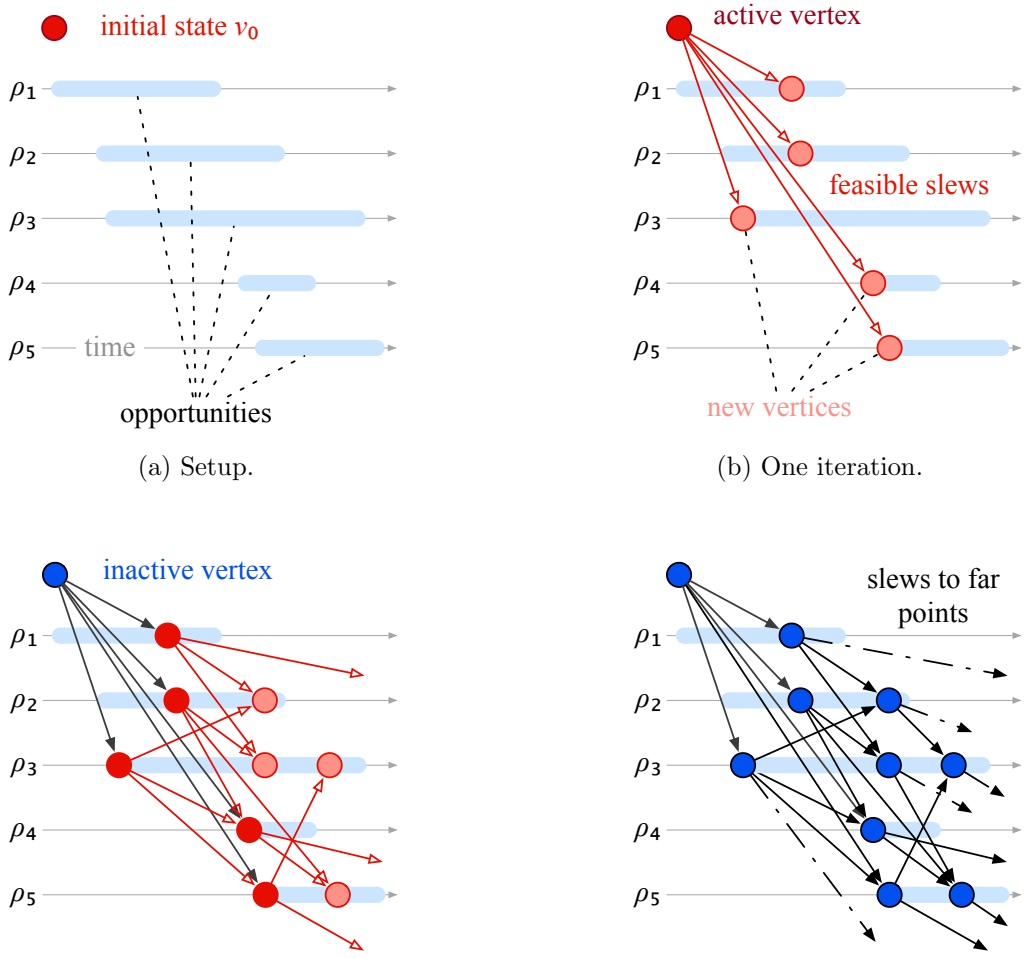
4.4.1 MILP-D: Dense Graph

MILP-D generates and solves a dense graph of possible slews between requests, representing the naïve approach to the problem.

4.4.1.1 Graph Construction

Algorithm 2 describes the process of recursively constructing a graph of all feasible slews for a satellite under the assumption of time-independent request values, shown in Figure 4.5. The algorithm begins with a single vertex v_0^s representing the satellite's initial state as parameterized by the network inputs in Equation 4.5. The transition time $\Delta t_{\text{slew}}(v_0^s, \cdot)$ is evaluated for all requests with opportunities in the upcoming planning horizon (if possible, leveraging efficient batchwise evaluation offered by most NN frameworks). If the estimated transition time indicates that the target is reachable before or during an opportunity window, a new vertex is added to the graph at the arrival time, clamped within the opportunity window. Note that only the earliest arrival from some v to some o must be included while preserving the optimality of the solution space for fixed-value requests, since opportunities reachable from ρ_n at t_i are a superset of opportunities reachable from ρ_n at $t_j > t_i$, with earlier departure times yielding the same or earlier arrival times. An edge with weight equal to the incoming target's value is added between the new vertex and the initial vertex. Any new vertices added this way are marked active A. The process is repeated for each active vertex until no new vertices are added. The maximum-weight path starting at v_0^s that avoids request duplication is the optimal sequence of requests to fulfill.

The design of the NN (i.e. such that the time-request tuples recorded at each vertex can be



(a) Setup.

(b) One iteration.

(c) Two iterations.

(d) Many iterations.

Figure 4.5: Dense slew feasibility graph construction using Algorithm 2.

used as the estimator's inputs) keeps the dimensionality of the graph low while retaining as much information about the dynamics as possible. If one were to include about-boresight rotation or other time-evolving quantities in the vertex information and network inputs, the dimensionality of the graph would increase exponentially with the planning horizon, making the problem intractable; the design allows vertices to only correspond to request and time (and not additional quantities such as complete attitude information). Furthermore, visitation times are discretized to some δt so that edges are formed between existing vertices, allowing the graph to "reconnect" to itself. Without this discretization, $|V|$ would grow exponentially with the planning horizon.

4.4.1.2 Mixed-Integer Linear Program

The MILP used to solve the dense graph is straightforward. To enforce non-repetition, each request partition may only be visited by one satellite at most once. Formally,

$$\text{maximize} \quad \sum_{s=1}^{N} \sum_{i,j} w_{i,j}^{s} x_{i,j}^{s} \tag{4.8}$$

$$\text{subject to} \quad \sum_{h} x_{h,i}^{s} + b_i^s \geq \sum_{j} x_{i,j}^{s} \quad \forall\ i \tag{4.9}$$

$$\sum_{s=1}^{N} \sum_{v_j^s \in p_n} \sum_{i} x_{i,j}^{s} \leq 1 \quad \forall\ n \tag{4.10}$$

where

$$b_i^s = \begin{cases} 1 & \text{if } i = i_{\text{start}} \\ 0 & \text{else} \end{cases} \tag{4.11}$$

The objective function, Equation 4.8, aims to maximize the sum of values for traveled edges, recalling that the weight $w_{i,j}^s$ of edge $(v_i^s \rightarrow v_j^s)$ is equal to the reward for visiting v_j^s. The first constraint, Equation 4.9, is the in-out constraint. For each vertex, the number of outgoing connections must be less than (at the end of the sequence) or equal (at all other points) to the number of incoming connections. The one exception is at each satellite's initial vertex, where b_i^s is set to one to seed the graph. Since the graph is acyclic, the summation on the L.H.S. must be zero, reducing the

constraint to

$$1 = \sum_{j} x^{s}_{i_{\text{start}},j} \tag{4.12}$$

As a consequence, all other vertices are restricted to one incoming and one outgoing edge selected, producing a feasible sequence. The second constraint, Equation 4.10, enforces at most one visitation of each partition, preventing duplicate credit for a single request by ensuring that each request is in U when visited.

While it would be possible to account for time dependence with this formulation by applying the previously described method to the graph, this would lead to optimization problems of an unacceptable size. Peng [137] uses an even denser method as an optimal benchmark for time-dependent instances with small request sets, but notes that it becomes too large for a computer's memory in larger instances.

4.4.2 MILP-S: Sparse Graph

MILP-S generates and solves a sparse graph of possible slews between requests, reducing the size of the MILP when compared to MILP-D. Additional variables must be added to support the different graph topology.

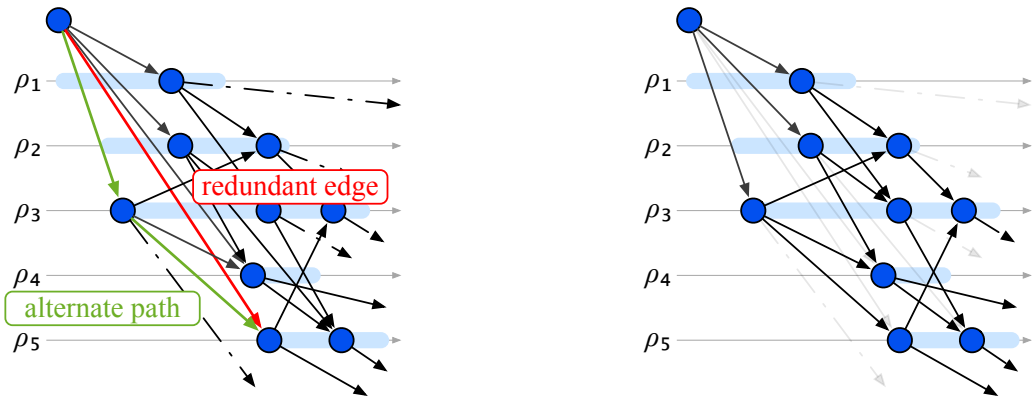
4.4.2.1 Graph Construction

Many of the edges in Algorithm 2 are redundant, as the feasibility of $v_i \to v_k$ can be inferred from the existence of $v_i \to v_j \to v_k$ through the composition of edges. Figure 4.6 demonstrates the removability of edges from the standpoint of transition feasibility.

Practically, generating the dense graph and deleting removable edges is computationally expensive. The generation of the sparse graph can be approximated if an upper bound on maximum transition time $\Delta t^{\max}_{\text{slew}}$ is known by modifying Algorithm 2 with Algorithm 3. For potential vertices J extending from v_i, only the vertices with t_j between the minimum $t_{j,\min}$ and one $\Delta t^{\max}_{\text{slew}}$ later need to be added to the graph; any subsequent vertices are guaranteed to be reachable from those

Algorithm 2 Dense slew graph construction for satellite s.

$V^s, A^s \leftarrow \{v_0^s\}$
$E^s, W^s \leftarrow \{\}$
for $v_i \in A^s$ **do**
  $J = \{\}$
  for $\rho_n \in R$ **where** $\exists\ o \in O_n^s$ s.t. $o_{\text{close}} > t_i$ **and** $\Delta t_{\text{slew}}(v_i, \boldsymbol{r}_n) \leq o_{\text{close}} - t_i$ **do**
    $t_j \leftarrow \text{clamp}(\text{ceil}(t_i + \Delta t_{\text{slew}}(v_i, \boldsymbol{r}_n), \delta t)), o_{\text{open}}, o_{\text{close}})$
    $J \leftarrow J \cup \{v_j = (t_j, \rho_n)\}$
  end for
  if sparse graph **then** execute Algorithm 3 **end if**
  for $v_j \in J$ **do**
    if $v_j \notin V$ **then**
      $V^s \leftarrow V^s \cup \{v_j\}$; $A^s \leftarrow A^s \cup \{v_j\}$
    end if
    $E^s \leftarrow E^s \cup \{(v_i \rightarrow v_j)\}$; $W^s \leftarrow W^s \cup \{r_j\}$
  end for
  if time-dependent value **then** execute Algorithm 4 **end if**
  $A^s \leftarrow A^s \setminus \{v_i\}$
end for



(a) Dense graph.

(b) Sparse graph.

Figure 4.6: Comparison between the dense graph with one removable edge highlighted and the sparse graph.

Algorithm 3 Sparse graph modification for Algorithm 2.

$t_{j,\text{min}} \leftarrow \min(t_j$ **for** $(t_j, \rho) \in J)$
for $v_j \in J$ **do**
  if $t_j > t_{j,\text{min}} + \Delta t_{\text{slew}}^{\text{max}}$ **then**
    $J \leftarrow J \setminus \{v_j\}$
  end if
end for

in the interval. While this does not produce the sparsest possible graph, it reduces $|E|$ by a factor of $\mathcal{O}(|\mathcal{R}|)$ at low computational cost.

4.4.2.2 Mixed-Integer Linear Program

Unlike the previous formulation, the optimization problem over the sparse graph is not a constrained path maximization problem, as solutions may pass through an already-visited partition without needing to access other regions of the graph. To handle this, an additional binary optimization slack variable y_n is added for each partition p_n representing whether r_n is fulfilled across all satellites:

$$\text{maximize} \quad \sum_{n} r_n y_n \tag{4.13}$$

$$\text{subject to} \quad \sum_{h} x^s_{h,i} + b^s_i \geq \sum_{j} x^s_{i,j} \quad \forall\ i \tag{4.14}$$

$$y_n \leq \sum_{s=1}^{N} \sum_{v^s_j \in p_n} \sum_{i} x_{i,j} \quad \forall\ n \tag{4.15}$$

The objective function, Equation 4.13, maximizes the sum of request values for each partition visited. The first constraint, Equation 4.14, is the same in-out constraint as in the previous formulation. Equation 4.15 enforces that the fulfillment variable y_n can only be 1 if the partition p_n is visited at least once by any satellite. Solutions produced by this method are equivalent to those from the naïve method but result from a smaller MILP formulation, as listed in Table 4.1.

4.4.3 MILP-TD: Time-Dependent Values

A final variation of the MILP is presented to handle the problem variation in which request value is a function of fulfillment time.

4.4.3.1 Graph Construction

In cases where request values are time-dependent, only including the earliest arrival time for each request in the graph is insufficient, as vertices may not be added or reachable for high-value

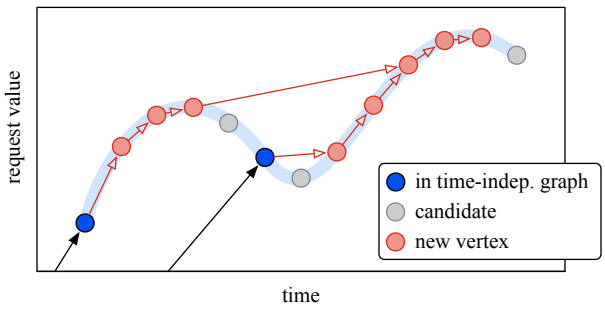


Figure 4.7: Slew graph with vertices added for time-dependent rewards.

Algorithm 4 Time-dependent request value modification for Algorithm 2.

for $t \in [t_i : \delta t : o_{\text{close}}]$ **do**
  if $r_i(t) > r_i(t_i)$ **then**
    $v_j \leftarrow (\rho_i, t)$
    if $v_j \notin V$ **then**
      $V \leftarrow V \cup \{v_j\}$
      $A \leftarrow A \cup \{v_j\}$
    end if
    $E \leftarrow E \cup \{(v_i \rightarrow v_j)\}$
    $W \leftarrow W \cup \{r_i(t)\}$
    break from iteration
  end if
end for

request times. For each vertex, an additional vertex is created if a higher-value fulfillment time follows in the opportunity window (up to discretization δt) as illustrated in Figure 4.7; candidate vertices that do not increase reward are ignored. Algorithm 4 describes the modification to Algorithm 2 to account for time-dependent rewards. This method is compatible with both the dense and sparse graph construction methods, but only the latter combination is considered so that the problem remains tractable.

4.4.3.2 Mixed-Integer Linear Program

In addition to the sequence decision variable $\boldsymbol{x}$ (and instead of $\boldsymbol{y}$ from the previous section), a binary optimization variable $z^s_{i,j}$ is added for each $x^s_{i,j}$, representing credit for visiting v^s_j. The problem is constrained to only allow credit to be assigned for a single visitation of each partition:

$$\text{maximize} \quad \sum_{s=1}^{N} \sum_{i,j} w^s_{i,j} z^s_{i,j} \tag{4.16}$$

$$\text{subject to} \quad \sum_{h} x^s_{h,i} + b^s_i \geq \sum_{j} x^s_{i,j} \quad \forall\ i \tag{4.17}$$

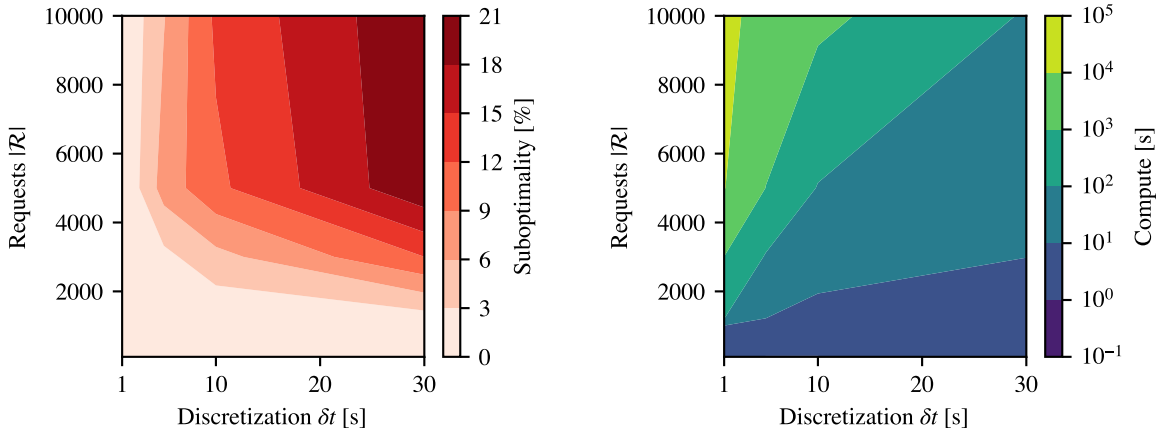
$$z^s_{i,j} \leq x^s_{i,j} \quad \forall\ i,j \tag{4.18}$$

$$\sum_{s=1}^{N} \sum_{v^s_j \in p_n} \sum_{i} y^s_{i,j} \leq 1 \quad \forall\ n \tag{4.19}$$

The objective function, Equation 4.16, maximizes the sum of credited request visitations. Equation 4.17 is the familiar in-out constraint. Equation 4.18 restricts credit assignment to edges that are visited. Finally, the constraint in Equation 4.19 only allows for credit to be assigned once per partition for any satellite.

4.5 Ablation Studies

Parameters in the MILP formulations are evaluated. An ablation study over δt is performed and the performance of MILP-S over the naïve MILP-D and a linear transition model are compared.



(a) Suboptimality due to discretization.

(b) Compute time.

Figure 4.8: Ablation study of MILP-S over time discretization δt.

4.5.1 Ablation Study on Time Discretization

Figure 4.8 shows the impact of the time discretization δt on the cumulative value of the optimal sequence of requests over one-orbit horizons, with a uniform request distribution. The value of the sequence decreases as δt increases, especially in high request density scenarios. The discretization $\delta t = 10$ seconds is selected for further studies, balancing the tradeoff between compute time and solution quality. In the densest cases ($|\mathcal{R}| = 10000$), this leads to a discretization error of $< 10\%$, while there is effectively no discretization error in sparse cases ($|\mathcal{R}| \leq 2000$).

4.5.2 MILP Formulation Comparison

The relative sizes and compute times of the naïve MILP-D formulation and the condensed MILP-S formulation are compared in Figure 4.9 over a small range of uniform request densities and horizons. The order of the number of edges in the graph is reduced by a factor of H (see Table 4.1) and some large constant in the sparse formulation, corresponding to a reduction in compute time. Extrapolating the trends indicates that MILP-D becomes intractable significantly more quickly than MILP-S. Beyond the time benefits of MILP-S, the relatively low memory requirement also allows for greater scalability.

4.5.3 Comparison to Linear Transition Model

A primary claim of this work is that the use of a NN to estimate transition times is more effective than a lower-order model. To evaluate this, the NN is replaced with a linear model of the form

$$\Delta t_{\text{slew}} = k\delta\theta \tag{4.20}$$

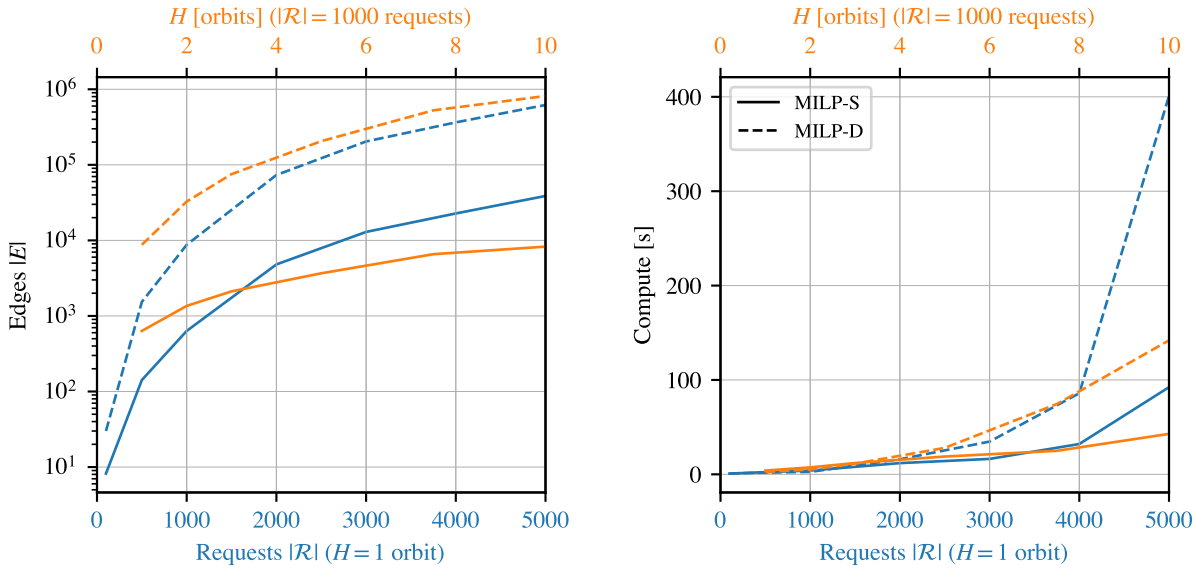
This model is evaluated over a range of k and $|\mathcal{R}|$ and compared to the NN-based solution for 1-orbit horizons. Figure 4.10 gives the results of this study. Varying k shows expected behavior: too low of a k leads to many failed slews (i.e. a large gap between planned and fulfilled requests), while too high of a k leads to significant idle time; in both cases, the overall reward is depressed. Since the transitions are nonlinear with respect to $\delta\theta$ and other variables, even the best value of k achieves a lower cumulative value than the NN-based solution because never are all slews accurately predicted; also, different values of k are best for different request densities. This effect is especially prominent as the request density grows and correct planning becomes more important. NN evaluation only accounts for a small portion of the graph construction time, so using it instead of a linear model does not significantly impact the overall compute time.

4.6 Performance

Benchmarks are performed over a range of request densities, distributions, and horizons for single satellite, time-dependent value, and multi-satellite cases. All MILPs are solved using the Gurobi solver [145].

4.6.1 Single Satellite Performance

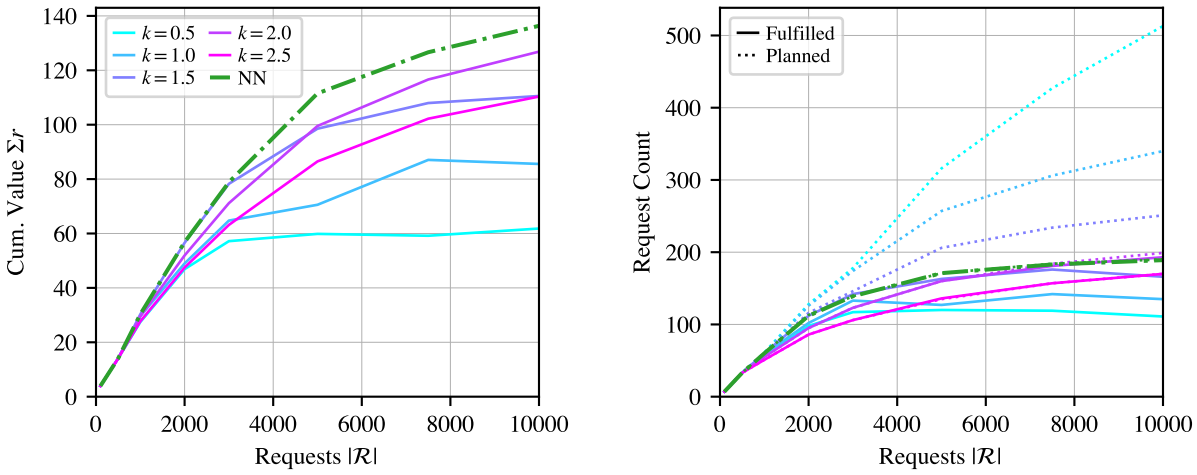
MILP-S is benchmarked over uniform and city request distributions, varying $|\mathcal{R}|$ and H with multiple trials at each point, for a total of 15 orbits worth of trials completed at each density. The results are collected in Figure 4.11 and Figure 4.12. The horizon-normalized cumulative value of the sequence $\Sigma r/H$ gives the value of the best MILP solution found and—in cases where the



(a) Graph size.

(b) Compute time.

Figure 4.9: Comparison of MILP-D and MILP-S over varied horizon H and request count $|\mathcal{R}|$.



(a) Cumulative value of fulfilled requests.

(b) Number of planned requests fulfilled.

Figure 4.10: Performance with a linear transition model with varying k versus the slew estimator.

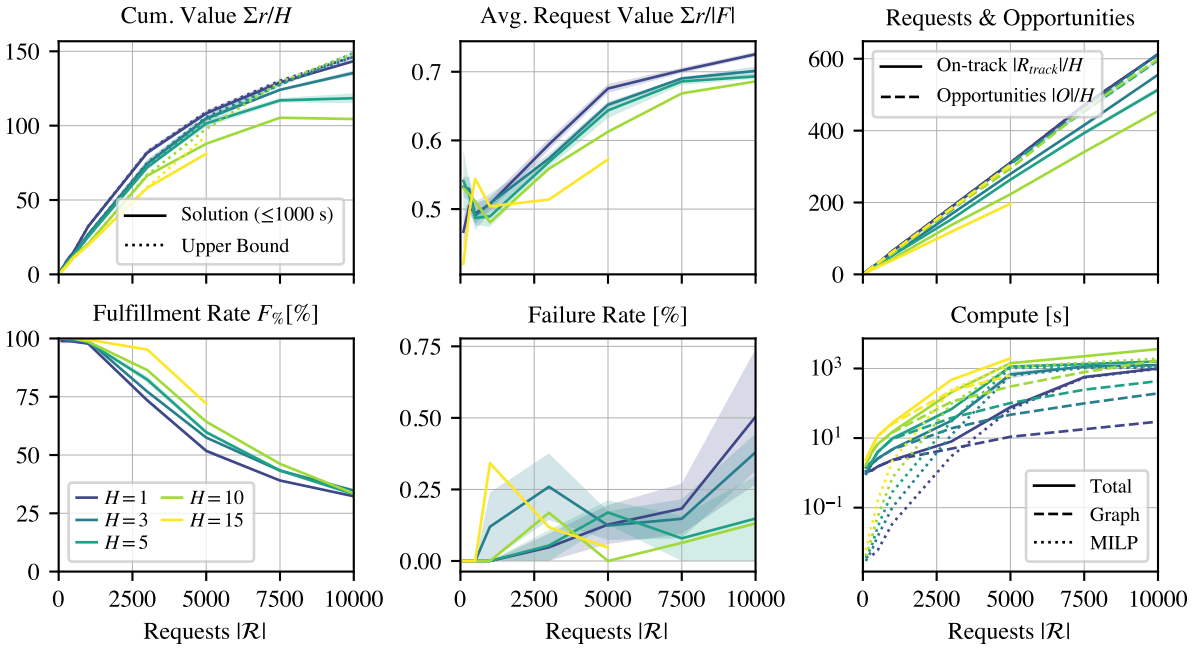


Figure 4.11: Benchmark of MILP-S on uniformly distributed, constant-valued requests. Shaded region corresponds to one standard error of the mean over trials.

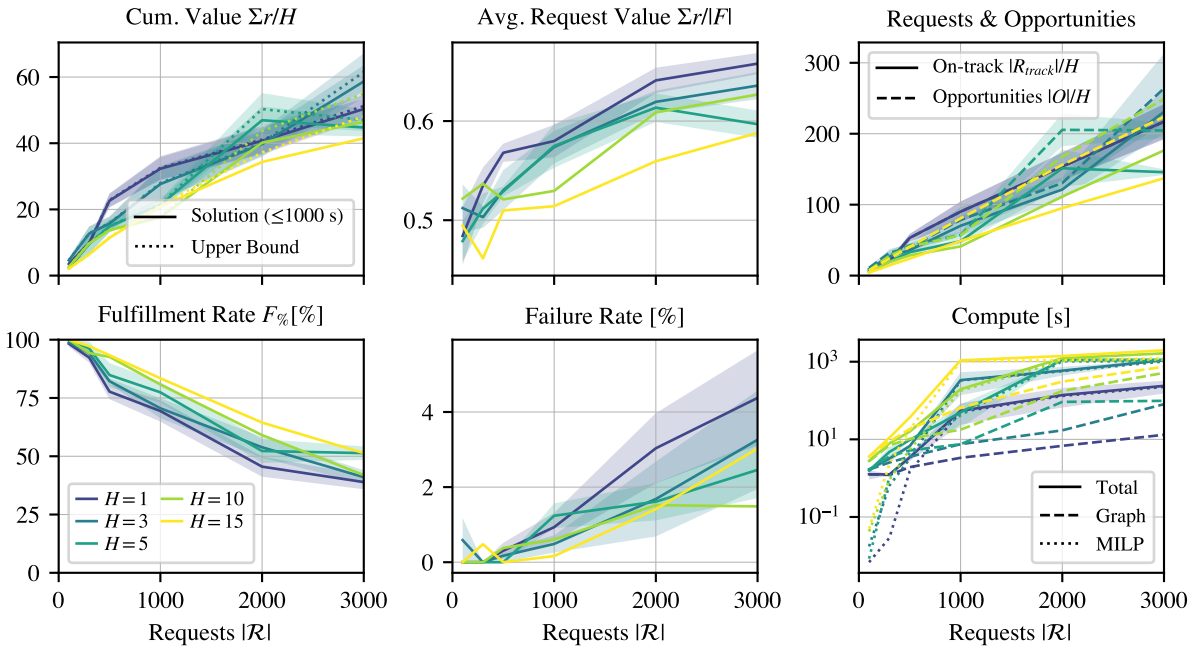


Figure 4.12: Benchmark of MILP-S on city-distributed, constant-valued requests. Shaded region as in Figure 4.11.

solver did not converge to optimality within 1000 seconds of solve time—an upper bound on the true solution value. Cases where the solver could not find any solution within 1000 seconds are excluded; practically, however, these cases tend to be those with the longest horizons and highest request densities so additional solve time could be allocated. The cumulative value tends to be lower for longer horizons, as the solutions are effectively less greedy; the satellite is competing with its future self for a limited number of requests. The average request value shows that as the number of requests increases, the satellite is able to be more selective about picking higher-value requests. The overall size of the problem is given in the requests and opportunities plot, showing the number the counts of each along-track over the horizon, giving insight into the size of the problem. The fulfillment rate $F_{\%}$ gives the ratio of requests fulfilled to the number of possible requests along-track; longer horizons and fewer requests give the satellite more opportunities to fulfill all requests. Finally, the success rate (plotted as its additive compliment, the failure rate) measures how effectively the MILP solutions can be executed in the high-fidelity simulation environment. It is given as the percent of requests successfully fulfilled in simulation out of the total planned requests by the MILP solver. The generally high success rates indicate that the solution pipeline is effective. Some depression in success rate is observed in high-density city-distributed cases as certain regions have extremely high local request densities, leading to an especially difficult problem instance.

4.6.2 Time-Dependent Priorities

Solutions for a single satellite with time-dependent rewards are benchmarked in Figure 4.13. Unsurprisingly, compute times tend to be worse than the fixed-value counterpart experiments due to the larger size of the graph; however, they are still tractable over a wide range of problem sizes. Despite many solutions showing a sizeable gap between the timed-out MILP solution and the upper bound, comparison to the constant-valued requests implies that the computed solution is likely close to the true optimum. The average request value is likewise lower since the satellite can only fulfill a request for maximum value if it is perfectly on-track. Still, the reduction is minimal,

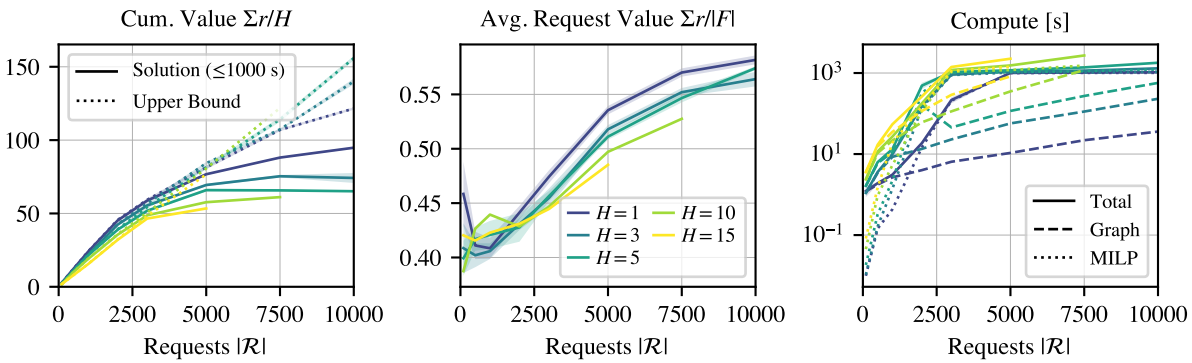


Figure 4.13: Benchmark of MILP-TD on the uniform request distribution with time-dependent priorities.

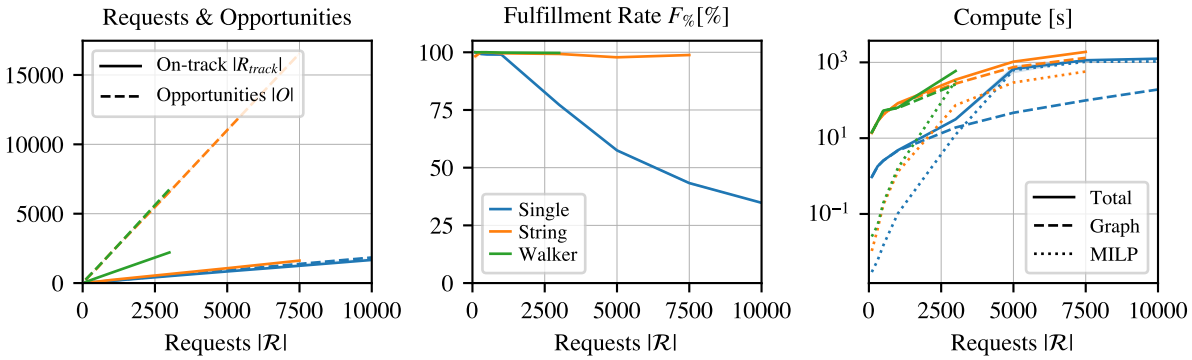


Figure 4.14: Benchmark of MILP-S on uniformly distributed, constant-value requests compared to a "String" constellation and a "Walker" constellation; $H = 3$ orbits.

indicating that the best possible imaging times are being chosen.

4.6.3 Multi-Satellite Scheduling

The performance of multi-satellite cases is examined in Figure 4.14. "String" and "Walker" constellations with $N = 12$ satellites, described in subsection 3.3.1, are tested. For the "String" case, satellites are separated by $\Delta f = 5°$, so satellites frequently share opportunities, as seen in the high number of opportunities and low number of on-track requests; the "Walker" case has less overlap between satellites.

As expected, both constellations are undersubscribed given the high number of overlapping

requests and the relatively high request fulfillment rate of a single satellite. Examining the compute times, the graph-construction time scales with N; however, each satellite's graph construction is independent and could be parallelized. A more interesting trend is observed in the MILP solution time (which is still capped at 1000 seconds for these experiments, hence the differing number of requests evaluated for each constellation): while the "Walker" constellation takes longer than the single satellite to solve as $|\mathcal{R}|$ increases, the "String" cases become easier to solve than the single satellite as $|\mathcal{R}|$ increases.

Further studies using the multi-agent MILP scheduler are examined in chapter 6, in comparison with other methods.

Chapter 5

Reinforcement Learning for the AEOSSP

5.1 Motivation

Preplanning approaches such as mixed-integer linear programs (MILPs) and iterative local search (ILS) solvers suffer from two major limitations: 1) high computational requirements and—as a result—slow solution times, especially as the problem size increases; and 2) being open loop, brittleness to a changing, uncertain, or mismodeled environment (in the case of MILP, the necessary linearization of resource dynamics guarantees some degree of mismodeling if included). If the plan is unsuccessfully executed midway through or more desirable objectives than those currently in the plan are added, either a new plan must be computed on the ground and reuploaded to the satellite or—computation capacity permitting—the satellite may locally repair a short horizon of the upcoming plan [3]. A variety of approaches have been proposed for reactive imaging (i.e. selecting tasks onboard based on the current environment state) [146, 16].

In particular, reinforcement learning (RL) is a broadly applicable framework for closed loop autonomy [25]. Nazari demonstrates that RL is effective at solving general waypoint routing problems [67]. Applied directly to spacecraft tasking, Harris [68], Hadj-Salah [59], and Eddy [28] formulate various Markov decision processes (MDPs) for the agile Earth-observing satellite scheduling problem (AEOSSP). References 28 and 59 use simplified probabilistic models for the spacecraft dynamics. In Reference 68 and later References 58, 63, 69, Harris and Herrmann develop the problem with a high-fidelity simulation and utilize shields to ensure operational safety. Additional work proves the maintenance of these safety conditions using formal methods [56]. Across these

formulations, fixed decision intervals are used, which can lead to suboptimal solutions when the satellite is capable of completing tasks in variable durations [68, 59, 63, 69]. Eddy formulates the problem as a semi-Markov decision process (sMDP) that can be solved by a Monte Carlo Tree Search algorithm that accounts for transition time costs [28].

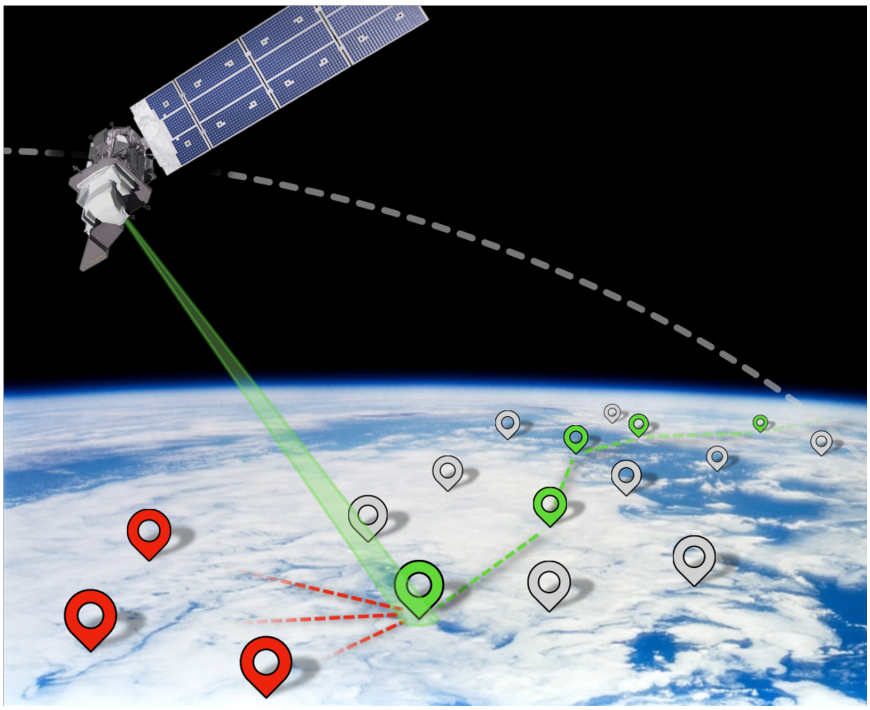


Figure 5.1: Concept figure of a satellite selecting which request to fulfill next.

In this chapter, autonomous agents are demonstrated executing learned policies with variable-duration decision intervals for the AEOSSP that are competitive with MILP-based approaches when compared in a high-fidelity simulation. Figure 5.1 shows a satellite considering which upcoming request to fulfill. Prior literature only provides formulations of the MDP or demonstrates solutions over simplifications of the problem (such as fixed decision intervals or graph-based abstractions). To learn an efficient policy on a high-fidelity instance of the problem, the problem is formulated as a sMDP in which actions are taken only for a minimum necessary duration to avoid satellite down-time. Furthermore, cases that the MILP planner from chapter 4 cannot handle, such as nonlinear spacecraft power resource constraints, are solved with RL. These RL-based solutions enjoy low

onboard computation costs, closed-loop opportunism, generalization across request distributions, and robustness to mismodeling, as opposed to the brittle, high-cost solutions provided by other methods.

The quality of the learned policies is evaluated across a variety of ablation studies. In the power-free environment, two sMDP formulations are shown to be superior to the MDP formulation, the performance of different observation spaces is compared, and the resulting policies are evaluated against the MILP-based solutions, demonstrating closed-loop performance that is competitive with the optimal global solution. Finally, the ability to learn policies for power-constrained environments is demonstrated, showing that policies from RL are able to contend with realistic and challenging constraints in a sensible manner. Overall, this chapter demonstrates in a high-fidelity simulation environment that RL is a viable and competitive method for solving the AEOSSP with onboard autonomy.

5.2 Markov Decision Processes with Variable-Duration Steps

As introduced in chapter 1, MDPs are a framework for defining sequential decision-making problems [25]. They consist of a state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability function $T(s'|s,a)$, reward function $R(s,a,s')$, and a discount factor γ. The agent observes the current state s_t and selects an action a_t based on a policy $\pi(a|s)$. The environment then transitions to a new state s_{t+1} according to the transition function, yielding a reward r_t. For a standard MDP, the goal of the agent is to learn the policy $\pi(a|s)$ that maximizes the expected value, or the γ-discounted sum of returns

$$V^{\pi}_{\text{MDP}}(s) = r + \gamma \mathbb{E}\left[V^{\pi}_{\text{MDP}}(s') \middle| T(s'|s,a), \pi(a|s)\right] \tag{5.1}$$

$$= r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = \sum_{i=0}^{\infty} \gamma^i r_i \tag{5.2}$$

where the discount factor $\gamma \in [0,1]$.[1] The latter expression (Equation 5.2) gives the value in terms of a sequence of experience tuples (s_t, a_t, r_t). However, this basic formulation assumes that the

[1] Typically, γ is close to 1.

opportunity-cost of each step is equivalent; in planning problems, this may not be the case.

5.2.1 Semi-Markov Decision Processes

SMDPs are a natural way of extending MDPs to steps with differing duration-costs. For an sMDP, a notion of continuous time is added to the problem formulation. This can be written as a probabilistic step-duration function $F(\Delta t|s, a, s')$ with a reward density ϱ instead of reward r [26], or as a MDP in which decisions are only made at certain supersteps, where each superstep consists of a variable number of substeps in the base MDP [147, 27], among other conventions. In general, the one-step γ-discounted reward for a general $\varrho_t(t)$ is given by

$$r_t^{(\gamma)} = \int_{t_t}^{t_t+\Delta t_t} e^{\gamma(t-t_t)} \varrho_t(t) \mathrm{d}t \tag{5.3}$$

In the sMDP context, γ is known as the discount rate with units of 1/time rather than the unitless discount factor.

If all reward for a step is distributed at the end of the step and is equal to r_t, the density function is

$$\varrho_t(t) = r_t \delta(t - (t_t + \Delta t_t)) \tag{5.4}$$

where δ is the Dirac delta function. As a result, further simplification is possible: experience tuples come in the form $(s_t, a_t, r_t, \Delta t_t)$, where Δt_t is the duration between s_t and s_{t+1}. The value function for this sMDP convention is given by

$$V_{\text{sMDP}}(s_0) = \gamma^{\Delta t_0} r_0 + \gamma^{\Delta t_0 + \Delta t_1} r_1 + \gamma^{\Delta t_0 + \Delta t_1 + \Delta t_2} r_2 + \dots \tag{5.5}$$

$$= \sum_{t=0}^{\infty} \gamma^{\sum_{i=0}^{t} \Delta t_i} r_t \tag{5.6}$$

and is used throughout this research.

5.2.2 Learning on an Infinite Horizon sMDP

The use of variable-duration decision intervals in the sMDP introduces challenges not typically encountered in RL, in which each step usually has an equal-time opportunity cost. Two methods

are considered to encode this: restructuring the sMDP as a MDP, and modifying the learning algorithm to discount according to the time spent at each step.

(1) The first method is to consider the problem for finite episodes, at the end of which no further rewards can be obtained. The time-through-episode is added to the observation, so that the agent can learn to treat time as an expendable resource. No modifications are needed to the learning algorithm, as the problem is no longer semi-Markovian.

This approach is not mathematically equivalent to the true sMDP formulation. However, as long as the problem has dynamics and rewards that are relatively uniform over an infinite horizon, learning to maximize reward in a sufficiently long fixed duration should optimize reward per time over all time.[2]

(2) Alternatively, the learning algorithm can be modified to discount by time instead of steps. In proximal policy optimization (PPO), the algorithm used in this work, the advantage estimator can be modified to account for the semi-Markovian nature of the problem. The standard form of the generalized advantage estimation (GAE) is given in Equation 1.7 as

$$\hat{A}^{\text{GAE}}_{t,\text{MDP}} = \sum_{i=0}^{\infty} \lambda^i \gamma^i \left(r_i + \gamma V(s_{t+1}) - V(s_t) \right)$$

By rederiving the GAE using the sMDP form of value (Equation 5.6), a modified GAE can be expressed as

$$\hat{A}^{\text{GAE}}_{t,\text{sMDP}} = \sum_{i=0}^{\infty} \lambda^i \gamma^{\sum_{j=0}^{i} \Delta t_{t+j}} \left(r_{t+i} + \gamma^{\Delta t_{t+i+1}} V(s_{t+i+1}) - V(s_{t+i}) \right) \tag{5.7}$$

Note that this convention matches the assumption that reward is received at the end of the episode. This formulation also differs from similar modified GAE in the RLlib examples [129] and in Menda et al. [27], as the exponentially decaying average is here taken over steps as opposed to duration.

[2] A MDP formulation that is equivalent to the sMDP is possible to construct, but not practical: By using the equivalency between a γ-discounted MDP and an undiscounted MDP with a $P = 1 - \gamma$ chance of termination at any step, the sMDP can be transformed into an undiscounted MDP with a state- and discount-rate-dependent chance of termination at each step.

With this modification, the agent can be trained in pseudo-infinite episodes, in which the episode is truncated after a fixed number of orbits and the value network is used to bootstrap estimates of additional value beyond that point. The inclusion of time-through-episode in the observation is not theoretically necessary in this case, but can still help with learning.

The impacts of these methods are compared for the AEOSSP in subsection 5.4.1.

5.3 MDP Formulation

The AEOSSP (as developed in chapter 3) for a single satellite can be represented as a partially-observable semi-Markov decision process (POsMDP). Actions correspond to high-level modes in the flight software (FSW), such as attempting to fulfill a certain request or—in the power-constrained variation—entering a charging mode. Variations of this mode-based approach to spacecraft RL were proposed by Harris et al., Eddy et al., and Herrmann and Schaub [68, 28, 69]. The agent is rewarded for fulfilling requests based on an operator-assigned per-request priority.

Much of the POsMDP—states, observations, and transitions—implicitly emerges from an underlying simulation of the environment. The POsMDP and underlying simulator is implemented using BSK-RL (see chapter 2). Ultimately, any effects that can be modeled in the simulation can be accounted for in the POsMDP and derived policies.

The POsMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, F, R, \mathcal{O}, Z)$. The simulator provides a deterministic generative model $G(s, a) = s'$. The elements of the partially-observable Markov decision process (POMDP) are defined as follows:

- **State Space $\mathcal{S}$:** The state space is the complete space of simulator states required to maintain the Markov assumption. This includes directly observable variables such as the spacecraft's dynamic state and upcoming request information, unobserved variables including far-off request information and request statuses, and variables required for simulation such as controller integrator states. Practically, only a subset of the state is relevant to the scheduling problem, which is exposed through the observation function.

			Observation Variant				
Quantity	**Norm.**	**Dim.**	**S1**	**S2**	**S3**	**P**	**Description**
$^{\mathcal{E}}\boldsymbol{\omega}_{\mathcal{BE}}$	0.03 rad/s	3	✓	✓	✓	✓	Body angular rate
$^{\mathcal{E}}\hat{\boldsymbol{c}}$	-	3	✓				Earth-frame instrument pointing direction
$^{\mathcal{H}}\hat{\boldsymbol{c}}$	-	3		✓	✓	✓	Hill-frame instrument pointing direction
$^{\mathcal{E}}\boldsymbol{r}_{\mathcal{BE}}$	r_E	3	✓	✓	✓	✓	Earth-fixed position, Earth radius-normalized
$^{\mathcal{E}}\boldsymbol{v}_{\mathcal{BE}}$	v_{orb}	3	✓	✓	✓	✓	Earth-fixed velocity, orbital velocity-normalized
r_n	-	N	✓	✓	✓	✓	Rewards $\in [0, 1]$ of next N requests
$^{\mathcal{E}}\boldsymbol{r}_{n\in N}$	r_E	$3N$	✓				Earth-frame positions of next N requests
$^{\mathcal{H}}\boldsymbol{r}_{n\in N}$	r_E	$3N$		✓	✓	✓	Hill-frame positions of next N requests
$\delta\theta_{n\in N}$	$\pi/2$ rad	N			✓	✓	Pointing error to next N requests
$\delta\omega_{n\in N}$	0.03 rad/s	N			✓	✓	Pointing rate error to next N requests
$t^o_{n\in N} - t, t^c_{n\in N} - t$	300 s	$2N$			✓	✓	Opportunity open and close of next N requests
t	$t_{\max}$	1	(Finite horizon MDP)				Time through episode (completion fraction)
Equation 5.9	5.0	K				✓	Upcoming request density
z	$z_{\max}$	1				✓	Battery charge fraction
t^o_{Ecl}, t^c_{Ecl}	T	2				✓	Next eclipse transitions, orbital period-normalized
$^{\mathcal{H}}\hat{\boldsymbol{s}}$	-	3				✓	Hill-frame Sun direction

Table 5.1: Elements in the observation o and their normalization constants.

- **Observation Space $\mathcal{O}$ and Observation Probability Function Z:** The observation space $\mathcal{O}$ consists of a selection of dimensions from the state space and transformations thereof. The selected elements are those presumed to be relevant to decision-making for the problem, based on expert knowledge and experimentation. Experiments are run for a variety of observations, given in Table 5.1. In general, these observations consist of some information about the spacecraft and some information about each of the N next upcoming unfulfilled requests. The request observations are ordered by their position along-track, making the assumption that nearby requests are most relevant for decision-making. The observation consists of information that the satellite would reasonably have available onboard with minimal uncertainty allowing for closed-loop operation onboard, other than updates to the request list from the ground. Observation elements are normalized to fall approximately in $[-1, 1]$, which improves the performance of deep reinforcement learning (DRL) algorithms. The number of upcoming unfulfilled requests in the observation N is tuneable. Observation elements are taken directly from the state without noise, though other papers have shown that reasonable noise can be added to the observation without impacting performance as long as the noise was modeled in training and deployment [84]. With the observation expressed as a subset function of state $O(s) \subset s$, the observation probability function is deterministic.[3]

  Observation S1 gives agent attitude and request locations in the planet-fixed frame, while S2 and S3 give those values in the agent-relative Hill frame. S3 also includes additional information about the angles to requests and access times.

- **Action Space $\mathcal{A}$:** The satellite has N imaging actions $a_{\text{im},n}$ that task the satellite with fulfilling the n^{th} upcoming unfulfilled request. As in the observation space, the upcoming requests are ordered by their position along-track and are recomputed at each decision interval, making the assumption that the agent would never reasonably want to select a

[3] Notated in the traditional manner, the observation probability function would be $Z(O(s)|s) = 1$.

far-off request as the next task. As shown in the results section, N must be balanced to give the agent sufficient information and choice without facing performance challenges associated with a very large action and observation space. Formulating the observation and action spaces in this manner (parameterized by N) is also conducive to RL: it maintains a reasonably small dimensionality for the observations and actions, and keeps those spaces a constant dimension, which is necessary for most DRL algorithms. The availability of an action does not imply feasibility: a request's opportunity window may close before the satellite can slew to and settle the instrument in the pointing direction in the high-fidelity simulation of the attitude control system. Since feasibility is not known *a priori*, an unfeasible action may be tasked and the controller will attempt to complete it until one of the criteria in the transition probability function is met.

- **Transition Probability Function $T(s'|s,a)$ and Step Duration Function $F(\Delta t|s,a,s')$:** Transitions are deterministic and generated by the simulator, resulting in $T(G(s,a)|s,a) = 1$. The simulator propagates for a non-constant duration at each step, depending on what conditions are deterministically met as the action is executed. For imaging actions, the simulator stops as soon as the satellite is successful or cannot be successful, thereby eliminating idle time. When generating the next state, the simulator runs until one of four conditions is met:

  (1) **Request Fulfilled:** If the request is successfully fulfilled (moved from $\mathcal{U}$ to $\mathcal{F}$), the step ends. No additional reward can be obtained from continuing to image the now-fulfilled request, so the satellite should retask immediately.

  (2) **Opportunity Close:** If the opportunity for the request being imaged closes before successful fulfillment, $t > t_n^c$, the step ends. There is no potential for the satellite to fulfill the request until a later orbit, so the satellite should retask immediately.

  (3) **Maximum Step Duration Timeout:** If neither of the above conditions are met before $\Delta t_{\max} = 5$ minutes have elapsed, the step ends.

(4) **Maximum Episode Duration Timeout:** If the maximum episode duration (5 orbits in training, as specified case-by-case for evaluation) is met, the step ends. Actions extending beyond the episode duration are interrupted.

The transition time Δt is deterministically the time elapsed between the previous state and the next state, which is determined by the simulator, $F(t' - t|s, a, s') = 1$.

- **Reward Function $R(s, a, s')$:** The agent is rewarded for successfully fulfilling requests. The reward function yields the request priority if the request is fulfilled during the step and zero otherwise:

$$R(s, a_{\text{im},n}, s') = \begin{cases} r_n & \text{if } \rho_n \in \mathcal{U} \text{ and } \rho_n \in \mathcal{F}' \\ 0 & \text{else} \end{cases} \tag{5.8}$$

5.4 Ablation Studies

Ablation studies are conducted to determine the impact of the sMDP formulation, the observation space parameterization, the lookahead distance N, and the request distribution used in training on the performance of the RL agents.

5.4.1 sMDP Formulations

To illustrate the impact of the sMDP modifications to PPO, an ablation study is performed to compare the performance of the standard GAE on finite episodes, the modified GAE on finite episodes, and the modified GAE on infinite truncated episodes. These experiments use the S3 observation on the sequencing-only problem formulation. Figure 5.2 shows that the modified GAE (Equation 5.7, used in the two sMDP columns) is more effective at learning efficient request sequencing than the standard GAE (Equation 1.7, used in the third MDP column), which too strongly favors high-value requests. The time-inclusive, finite-horizon sMDP formulation is used throughout the rest of the chapter (arbitrarily selected due to equivalent performance to the pseudo-infinite horizon formulation).

5.4.2 Request Lookahead and Distribution

The second ablation study investigates how the request lookahead distance N and the training distribution impact performance. Six agents are trained for each combination of $N \in [16, 32, 64]$ and training distribution $\in$ [Uniform, Cities] using the S3 observation from Table 5.1. Once trained, the agents are evaluated on different densities of each distribution for a 10-orbit horizon. These results are compared to the MILP solution (found in $\leq$ 1 hour) and the MILP upper bound in Figure 5.3. The true optimality of the policy lies between the 1-hour solution and upper-bound lines, but practically the MILP solver will not have the solution and upper bound converge in a reasonable solve time for large instances of the problem. The action distributions of each agent in the evaluation cases and the equivalent actions selected by the MILP are given in Figure 5.4. Since lower index actions correspond to a request that is closer along-track, low-index actions tend to be selected more often. In the city request distribution, high action indices tend to be selected more often due to the clustering of requests, which sometimes leads to a large number of immediately upcoming requests that are infeasible to slew to and must be skipped.

When evaluated in new instances of the uniform request distribution environment, consisting of request sets not seen in training, all of the RL agents perform very well. At high request densities, every agent outperforms the MILP solver's 1-hour, open-loop solution using only about 10 milliseconds of computation per decision in a closed-loop manner. Even when compared to the upper bound for cumulative reward found by the MILP solver, which is a conservative bound, the RL agents collected 80% of the bound. Across lower request densities, the best agents never collect less than 90% of the MILP solution or bound. Beyond this, moving such MILP solvers to flight hardware would be challenging and less efficient.

The city-distributed evaluation environment is more challenging for the RL agents due to some periods with a high number of conflicting requests. Performance is better on lower request densities than higher densities, with many agents collecting 80% or more of the optimal reward. At high densities, the best agents, which were trained in the city environment, collect 77% of the

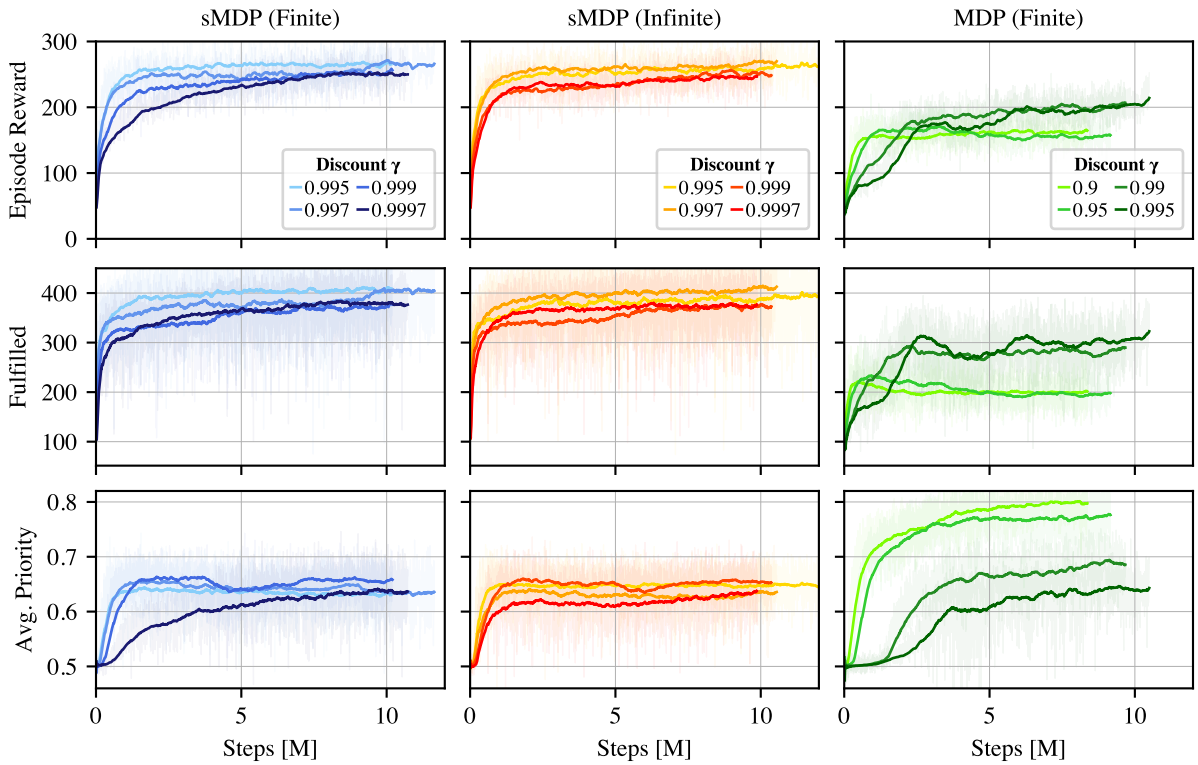
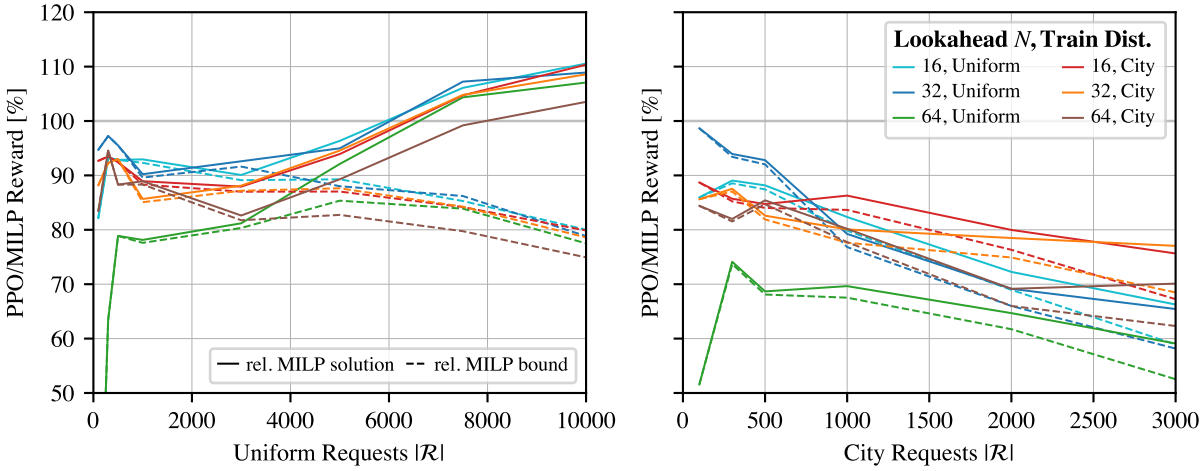


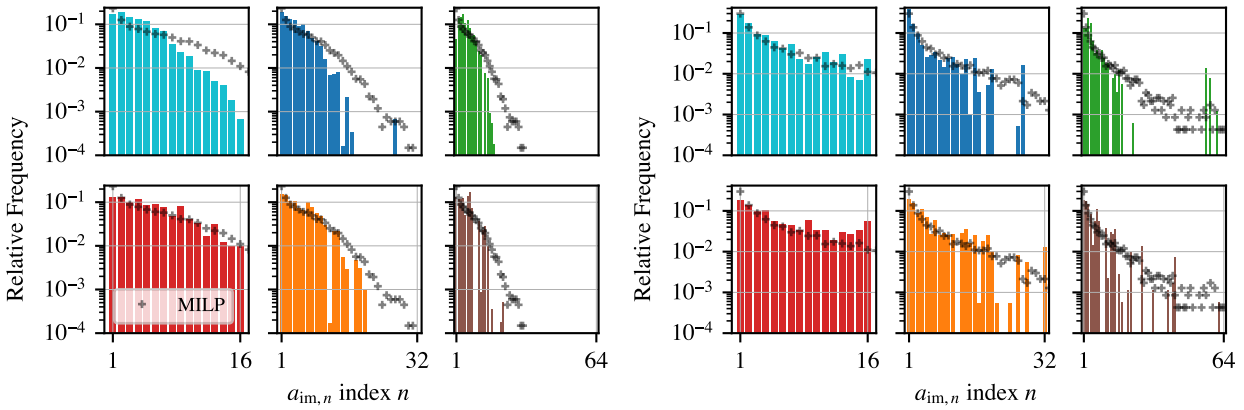
Figure 5.2: Comparison of the step- and time-discounted methods over a range of γ.



(a) Uniformly distributed request environment.

(b) City-distributed request environment.

Figure 5.3: Cumulative reward of RL agents compared to MILP solution and upper bound with up to 1 hour of MILP time.



(a) Uniformly distributed request environment.

(b) City-distributed request environment.

Figure 5.4: Relative frequency of policy actions compared to equivalent MILP solution actions.

MILP solution and 68% of the upper bound.

Agent performance is affected by the lookahead N, as seen in Figure 5.3. If N is too small, necessary information and optimal actions may be truncated from their respective spaces; this is visible in Figure 5.4, which gives the relative frequency of each action executed by the policy. The MILP solution is seen to select actions up to $a_{\text{im},32}$ in the uniform environment and above $a_{\text{im},64}$ in the city environment; the agents trained to consider only $N = 16$ upcoming requests are thus unable to mimic the selections of the MILP solution, particularly in the city case. However, too large an N can lead to extraneous information in a dimensionally large observation space, which impacts the ability of the agent to learn, leading to depressed cumulative rewards in some $N = 64$ cases. In general, the $N = 32$ cases strike a balance between the competing effects.

The selection of training request distribution is a less significant factor in overall performance. For both the uniform and city request cases, the total number of requests is randomized each episode in training. As a result, the agent experiences a large range of local densities, allowing for generalization to cases where the request density is non-homogeneous. Since the S3 observation parameterizes request information to be relative to the agent, the fact that the city requests do not cover the entire domain of Earth's surface does not significantly impact the ability of the agent to generalize to a globally distributed set of requests because the agent has learned throughout the

entire relative domain. Still, there is a slight preference of the agents to be deployed in the same distribution that they were trained with.

5.4.3 Observation Parameterization

This ablation study compares the performance of agents trained with different observation parameterizations. An agent is trained for 24 hours on a uniform request distribution with $N = 32$ for each of the S1, S2, and S3 observations given in Table 5.1. Notably, the observations are computable from one another's elements, so this study tests whether different representations of the same information can improve the agent's learning.

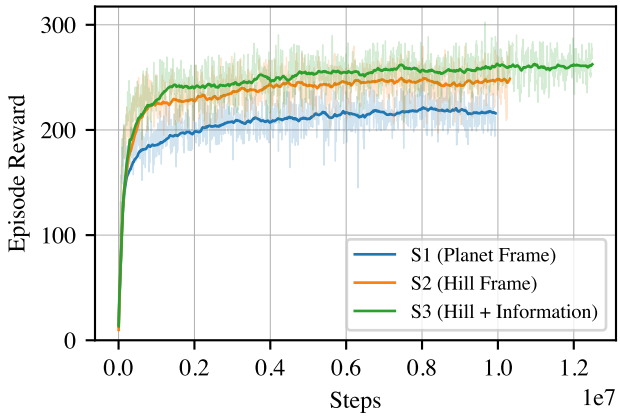


Figure 5.5: Observation ablation study training curves.

The training curve for each agent is shown in Figure 5.5. The S3 observation outperforms the S1 and S2 observations in training, with the S1 observation performing about 20% worse than S3. Since the S3 observation more explicitly provides information needed to find a good policy, the network does not need to learn to extract that information from the state space. As a result, the expressivity of the policy network can be used more directly on tasking as opposed to implicitly learning the computations performed in S3.

5.4.4 Analysis of Best Policies

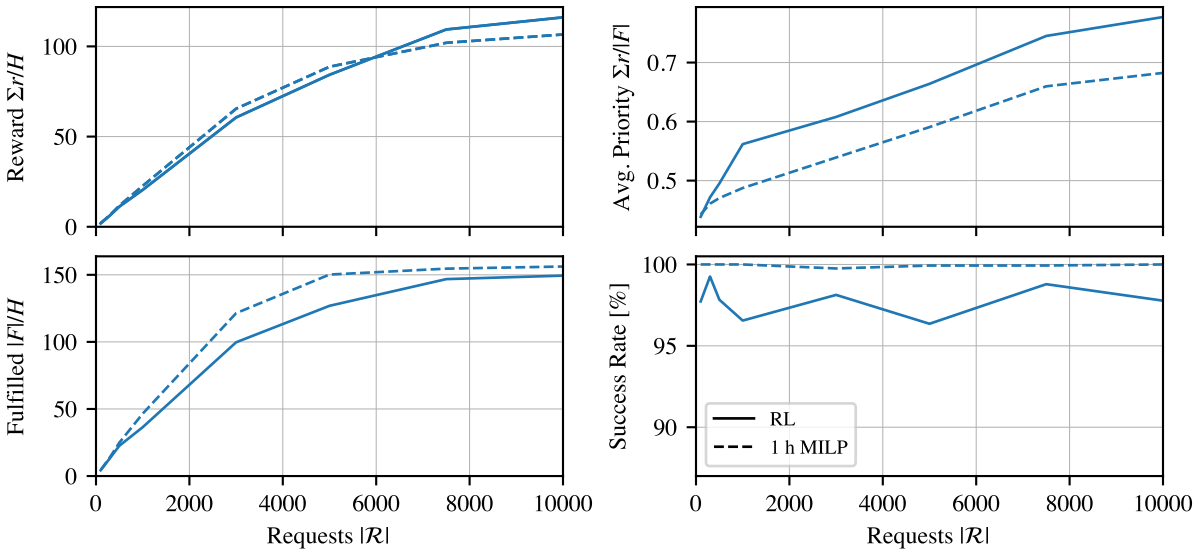
A closer analysis of the best policy for each request distribution is performed on a 10-orbit horizon, shown in Figure 5.6. The best policies use $N = 32$ with the S3 observation, trained on the uniform distribution for uniform evaluation and on the city distribution for city evaluation. These results are compared to the MILP-based solution.

For both request distributions, the fulfillment rate is lower but the average fulfilled request priority is higher for the RL agents compared to the MILP solution. Since the overall performance is similar, this implies that a range of strategies for the problem can be successful. It follows that the RL agents converge on a strategy that requires less optimization of long sequences of lower priority requests if a more direct policy can achieve similar value.

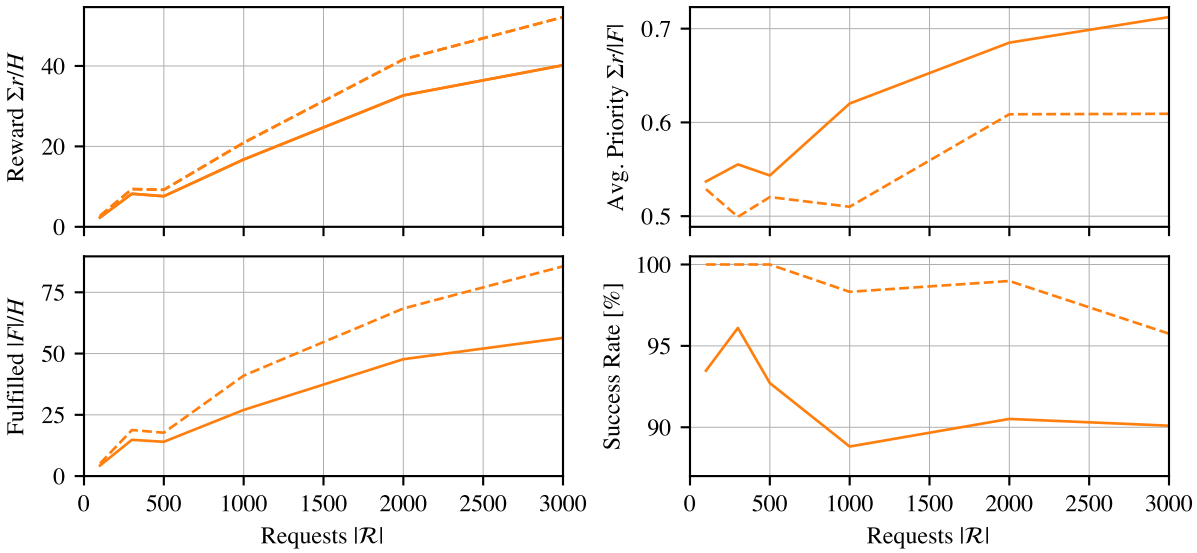
Looking at the success rate of the policies—i.e. what fraction of the attempted actions are completed successfully—is also elucidating. Actions will fail if the selected request cannot be slewed to before it is out of range. Compared to the MILP solution which only misses requests in cases where the simulated dynamics differ too much from the model, the RL agents miss requests at a higher rate. This is not inherently bad because there is no explicit penalty for an unsuccessful action other than the opportunity cost, but it does imply room for improvement. This is particularly evident in the city distribution, where the tight clustering of requests causes even the MILP solver to have a lower success rate. In high-density instances of this case, the RL agent misses as many as 10% of all attempted requests.

5.5 Power-Constrained Environment

The power-constrained AEOSSP defined in subsection 3.4.2 is also solved with RL. This demonstrates a key benefit of learning-based methods: Since it is possible to simulate the dynamics of this additional subsystem as part of the learning environment, accounting for its impacts can be achieved by training with the subsystem's models present in the simulator. In comparison, adding such constraints to other optimization frameworks that rely on a hand-crafted model may



(a) Uniform policy and evaluation environment.



(b) City policy and evaluation environment.

Figure 5.6: Best policy for each distribution over a range of request counts.

be difficult or impossible.

5.5.1 MDP Modifications

The AEOSSP POsMDP is modified in four places to allow the agent to monitor and respond to the constraints of the power system:

(1) **Simulation:** The underlying Basilisk simulation that gives state transitions is modified to include the power subsystem. While reformulating some planning algorithms to account for power is challenging, the POsMDP formulation allows for the straightforward inclusion of power constraints directly from the power dynamics model.

(2) **Observation Space:** As listed in Table 5.1 column **P**, the observation in the power-constrained variation is augmented with relevant information about the charge state. This includes current charge z, the eclipse transition times t^o_{Ecl}, t^c_{Ecl}, and wheel speeds, among other elements. To encode long-term presence or absence of upcoming requests, the request density function

$$\left\{ \sum_{\rho_i \in \mathcal{P}_k} r_i \;\middle|\; k = 1, \ldots, K \right\}, \quad \mathcal{P}_k = \{\rho_i \in \mathcal{U} \mid t + (k-1)\Delta t \leq t_i^o < t + k\Delta t\} \tag{5.9}$$

is added to the observation, computed from the satellite's onboard request list; it gives the cumulative priorities of unfulfilled requests in each of the next $K = 20$ intervals of $\Delta t = 5$ minutes. The intent of this observation is to reveal gaps in upcoming request availability without needing a very large N to expose a long horizon of requests to the agent. With this information, the agent should be able to identify if there are upcoming periods with fewer requests that could be used to perform resource management actions.

(3) **Action Space:** The satellite is given an additional charging action, a_{charge}, which points the solar array towards the Sun for 5 minutes. Charging is determined by the underlying simulation: Power is passively generated at any point when the solar panels are exposed to the Sun (even if $a \neq a_{\text{charge}}$). Conversely, tasking a_{charge} does not guarantee charging,

such as if the satellite is in eclipse. The charging action has no explicit reward or penalty associated with it; the agent must learn to use it to avoid failure states while accounting for the implicit opportunity cost of not imaging.

(4) **Reward Function:** If the satellite enters a discharged state where $z = 0$, the episode ends. This is a hard constraint that the agent must learn to avoid. A reward $r_{\text{penalty}} \leq 0$ is yielded for entering a failure state.

5.5.2 Simple Shield for Power Management

Shielded RL was introduced by Alshiekh et al. as a way of making safety guarantees about the safety of an RL agent [55]. Since the power-constrained formulation introduces a domain of unsafe states (i.e. states where $z \leq 0$), a shield can be used to prevent the satellite from entering those states.

A relatively simple hand-designed shield is employed. The shield evaluates the policy's chosen action against the observation; the shield either accepts the action as safe or overrides the potentially unsafe action with a safe action. Harris et al. apply this to spacecraft tasking to meet the safety requirements of space systems [58].

In particular, the shield overrides the selected action with the charging action if the battery level is below

$$z_{\text{minsafe}} = \begin{cases} z_{\text{floor}} - (t^c_{Ecl} - t^o_{Ecl})\dot{z}_{\text{draw}} - t^o_{Ecl}\dot{z}_{\text{gain}} & \text{if not in eclipse and } z > z_{\text{floor}} \\ z_{\text{floor}} - t^c_{Ecl}\dot{z}_{\text{draw}} & \text{if in eclipse} \\ z_{\text{floor}} & \text{else} \end{cases} \tag{5.10}$$

which maintains the battery level at least at z_{floor}, with a higher requirement as eclipse nears in order to survive eclipse without losing power. Worst-case estimates of $\dot{z}_{\text{gain}} = 100\%/\text{orbit}$ and $\dot{z}_{\text{draw}} = -100\%/\text{orbit}$ are used, and z_{floor} is set to 25%. While this shield is not optimal, it is effective at preventing the agent from entering failure states without being over-conservative.

While outside the scope of this dissertation, more sophisticated techniques for automatic shield generation with performance and robustness guarantees are being explored [105, 56, 57].

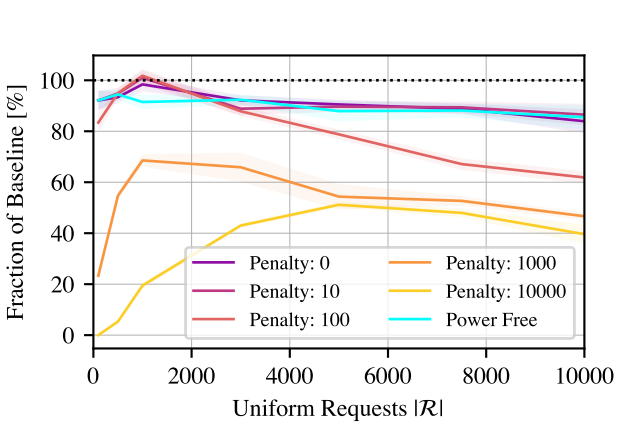
5.5.3 Experiments

For each distribution of requests, five policies are trained with different failure penalties in the power-constrained environment. Failure penalties are a negative reward returned when the agent enters a failure state, i.e. when the battery level reaches zero. The value of this penalty must be tuned, like any other hyperparameter.

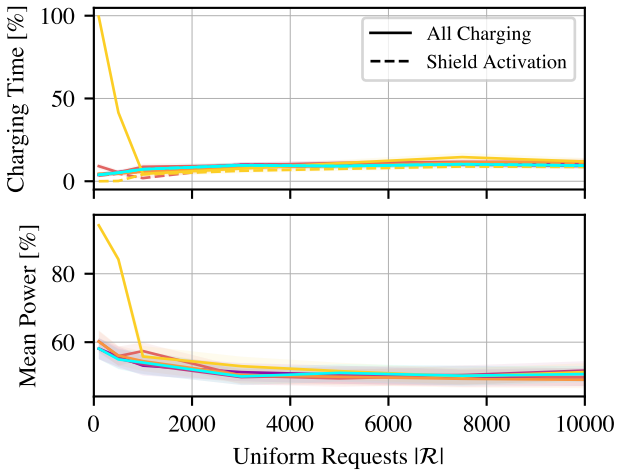
Following from prior results, it is assumed that the distribution of requests in the deployment environment is sufficiently well known to be able to train policies on similar distributions over a range of densities. This ensures that the power-constrained results are only considering the impact of the power subsystem on the agent's performance, not the impact of transferring between different request distributions.

In testing, the policies trained with different failure penalties are evaluated with the shield given in Equation 5.10 to ensure safety. An additional policy is trained in a power-free environment and deployed in the power-constrained environment with a shield; this provides a lower baseline for comparison, i.e. answering the question of how much better the other policies can perform if they are actively aware of power constraints. The results are compared to the baseline performance of a power-free policy in a power-free environment with the same request list, such that statistics are given relative to an agent that has no concern about power. Results are given in Figure 5.7 and Figure 5.8.

Considering the uniform request distribution (Figure 5.7), there is no intuitive way for the satellite to exploit the environment. Because requests are uniformly distributed, there is a low probability of a single 5-minute period having significantly less reward available than any other period. As a result, the shielded power-free policy performs nearly as well as the shielded power-constrained policies. An exception to this trend is in the lower request density regime (e.g. 500 to 3000 global requests), in which there are periods of low reward availability that the power-

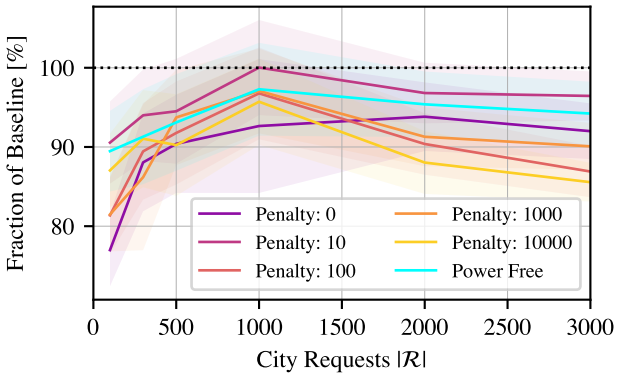


(a) Total reward relative to power-free baseline.

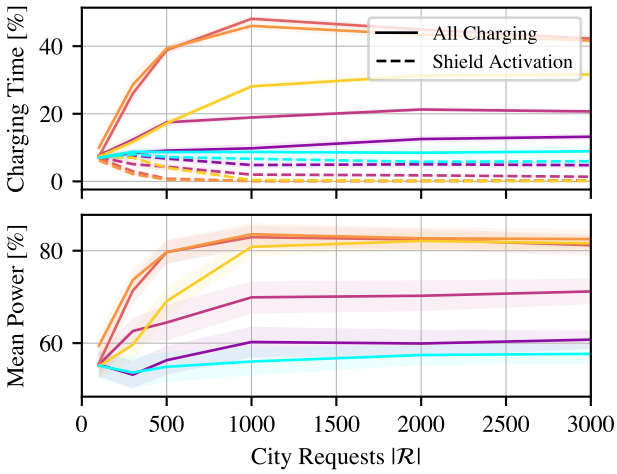


(b) Power subsystem statistics.

Figure 5.7: Benchmark on the uniform request distribution.



(a) Total reward relative to power-free baseline.



(b) Power subsystem statistics.

Figure 5.8: Benchmark on the city request distribution.

constrained policies proactively choose to charge in, as reflected by higher rewards. While training curves show that the power-constrained policies did tend to stay alive, they have learned to do so in a less conservative manner than the shield dictates, as most of their charge actions are forced by the shield on the uniform distribution.

Compared to the uniform distribution of requests, the city distribution is more exploitable. The city distribution has periods of high and low reward availability; while the shield has no awareness of rewards that may be neglected by charging, the power-constrained policies can learn to charge only in periods of low/no reward availability (e.g. over oceans and other sparsely inhabited areas). In Figure 5.8, the best power-constrained policy is shown to always outperform the shielded power-free policy; it often achieves performance close to the power-free baseline, which indicates that it has learned to charge in periods that never cause it to lose potential rewards. The power-constrained policies also maintain the spacecraft in a safer state, as reflected by higher average charge levels over the course of the simulations. While the shield only tries to prevent failure, the RL-based solutions attempt to maximize success, which includes maintaining a higher charge level to remain farther from failure states.

Chapter 6

Inducing Collaboration in Observation Constellations

6.1 Motivation

Scheduling for agile Earth-observing satellites (AEOSs) becomes more challenging when multiple agents are considered, resulting in the agile Earth-observing constellation scheduling problem (AEOCSP). Optimizing and exploiting multi-agent interaction has been identified as an important next step in multi-agent satellite planning for coordinated and reactive tasks [148]. Asynchronous inter-agent reactivity can be combined with existing allocation algorithms, as demonstrated in References 149 and 150. In these, modest performance improvements are seen since dynamic events are more effectively allocated. Holder takes the approach of modifying the training pipeline when learning in a multi-agent environment to directly enforce task deduplication [151]. To generalize to a scalable multi-satellite constellation using reinforcement learning (RL) in a basic tasking scenario, Reference 66 adds communication between single-agent-trained satellites to deduplicate efforts. However, no prior work combines scalability and performance in a high-fidelity environment, yielding operation-quality results.

This chapter considers the AEOCSP in a distributed, autonomous manner by modifying the information-sharing methods conceptualized in Reference 66 to allow for the use of single-agent RL-based policies in multi-agent instances of the high-fidelity agile Earth-observing satellite scheduling problem (AEOSSP). To enable this, methods for coordinating asynchronous decision-making agents from the previous chapter are developed to ensure task deconfliction and equitable division of labor among constellation members. The autonomous methods are compared to the mixed-integer linear

program (MILP)-based approach that yields globally optimal solutions, allowing the optimality of the autonomous method to be quantified. The RL-based method shows superior performance at a significantly lower computational cost over a range of problem sizes. The method is then applied to a large-scale case with dozens of satellites and tens of thousands of dynamically appearing requests to demonstrate scalability and performance.

6.2 Problem Formulation

This chapter continues to use the AEOSSP problem formulation given in chapter 3. Additional notation is introduced for sets of intended tasks, and the AEOSSP Markov decision process (MDP) is formalized for multiple satellites.

6.2.1 Intent Sets

While the AEOSSP problem formulation defines the unfulfilled $\mathcal{U}$ and fulfilled $\mathcal{F}$ request sets, the algorithms in this chapter require a third set to be defined. This is the intent set $\mathcal{I}$, containing requests that some satellite intends to—but is not necessarily guaranteed to—complete.

When a satellite attempts to fulfill some request ρ_i, the request is added to $\mathcal{I}$, while remaining in $\mathcal{U}$. As usual, if the task is successful, the request is moved from $\mathcal{U}$ to $\mathcal{F}$; if unsuccessful it remains in $\mathcal{U}$. In either case, the request is removed from $\mathcal{I}$ as soon as the satellite is no longer trying to complete it.

For the purposes of this chapter, it is assumed that satellites are able to communicate globally and instantaneously at will. As a result, all satellites have up-to-date information on the global sets $\mathcal{U}$, $\mathcal{F}$, and $\mathcal{I}$ at all times. While trends towards increased in-space communication suggest this assumption may be reasonable in the near future, Reference 66 implies that local communication in the constellation should be sufficient for good performance, as information about the actions of far-off satellites is mostly irrelevant for short-term planning.

6.2.2 Decentralized MDP Formulation

The MDP formulation of the AEOSSP defined in section 5.3 must be expanded to multiple agents. The multi-agent problem is formalized as a decentralized partially-observable semi-Markov decision process (Dec-POsMDP), a generalization of a MDP for decentralized decision-making between multiple agents with a single goal [152]. Note, however, that in this chapter, training is only performed on a single agent at a time. Since the original problem formulation is semi-Markov, each satellite's action duration is independently determined. As a result of there being multiple agents acting in a decentralized manner with different-duration actions, agents act asynchronously [27].

The elements of the Dec-POsMDP are as follows:

- **State Space $\mathcal{S}$:** As in the single-agent formulation, the space of simulator states required to maintain the Markov assumption. This consists of the satellite state spaces and the environment state, $\mathcal{S} = \mathcal{S}_1 \times ... \times \mathcal{S}_N \times \mathcal{S}_{\text{env}}$.

- **Per-Satellite Observation Space $\mathcal{O}_s$:** The observation for each satellite is the same as in the single-agent formulation, as given in Table 5.1. When determining which upcoming requests to include in the observation, they are selected either from $\mathcal{U}$ or $\mathcal{U} \setminus \mathcal{I}$ depending on the collaboration method being used.

- **Per-Satellite Action Space $\mathcal{A}_s$:** The actions available to each satellite are the same as in the single-agent formulation: each satellite has $N = 32$ actions $a_{\text{im},n}$ corresponding to imaging each of the requests represented in the observation.

- **Transition Function:** Transitions and step durations are given by the generative model $G(s, a) = s'$ from BSK-RL. The simulator propagates for a variable amount of time, depending on the actions taken and how the environment evolves during propagation. Propagation is halted when one of three conditions is met for any satellite(s):

  (1) **Imaging Successful:** If any satellite tasked with an imaging action successfully

images the target, the simulation halts since that satellite's action cannot result in any more reward.

(2) **Opportunity Window Close:** If the opportunity window closes for any satellite tasked with an imaging action, the simulation halts since the action will not result in any reward.

(3) **Maximum Step Duration Timeout:** If neither above condition is met for the satellite after $\Delta t = 5$ minutes of simulation time since the last time a new task was assigned, the simulation halts.

The simulator returns the next state s', the reward r, and the step duration Δt. The satellite(s) for which the simulation was stopped require a new action to be assigned at this point, but other satellites may retask if desired, beginning from the interrupted mid-task state.

- **Reward Function $R(s, s')$:** As in the single-agent formulation, the agents are jointly rewarded for imaging unfulfilled requests during a step

$$R(s, s') = \sum_{\rho_i \in \mathcal{R}} \begin{cases} r_i & \text{if } \rho_i \in \mathcal{U} \text{ and } \rho_i \in \mathcal{F}' \\ 0 & \text{else} \end{cases} \tag{6.1}$$

where $\mathcal{U}$ are the unfulfilled requests at s and $\mathcal{F}'$ are the fulfilled requests at s'. Since the task is collaborative, it does not matter which agents get credit for a particular reward.

6.3 Algorithms for Collaboration

Satellites are trained in a single-agent environment, but must be induced to work collaboratively in a multi-agent setting. This is achieved by communicating the statuses of requests within the constellation as agents make progress toward the joint goal. These algorithms induce collaboration and resolve conflicts among agents in the multi-agent setting.

6.3.1 Algorithm Architecture

Algorithm 5 presents the general algorithm used for determining joint actions. First, the set of satellites $\mathbb{S}_R$ that will receive a new task at the current step are identified depending on the definition of NeedRetasking. Next, the sets of fulfilled, unfulfilled, and intended requests are updated, based on the satellites' Task from the previous step and whether TaskSuccessful. (Algorithm 6). Satellites that are not retasking at the current step retain their intents from previous steps. In practice, this can be achieved by having all satellites broadcast when they have successfully fulfilled a request or abandoned an attempt; the status sets can be updated locally onboard each satellite with this information. The set of requests $\mathcal{R}_{\text{avail}}$ that retasking satellites may choose from is also determined depending on the method being used for AvailableRequests.

With the satellites and their options identified, the policy can be evaluated for each satellite. Each satellite's observation is obtained by first identifying the next N UpcomingRequests, then computing the per-satellite Observation o_s as defined in Table 5.1 column S3. As opposed to selecting an imaging action $a_{\text{im},n}$ directly, the policy's probability distribution across requests is saved to P_A. Finally, a method to SelectActions can be applied to the probabilities to map from retasking agents to requests.

Algorithm 5 Generalized joint action computation function

function ComputeJointAction($\mathbb{S}$; NeedRetasking, AvailableRequests, SelectActions)
  $\mathbb{S}_R \leftarrow$ NeedRetasking($\mathbb{S}$)
  $\mathcal{U}, \mathcal{F}, \mathcal{I} \leftarrow$ UpdateRequestSets($\mathbb{S}, \mathbb{S}_R, \mathcal{U}, \mathcal{F}, \mathcal{I}$)
  $\mathcal{R}_{\text{avail}} \leftarrow$ AvailableRequests($\mathcal{U}, \mathcal{I}$)
  $P_A \leftarrow \emptyset$
  for $s \in \mathbb{S}_R$ **do**
    $\mathcal{R}_{\text{upcoming}} \leftarrow$ UpcomingRequests($s, \mathcal{R}_{\text{avail}}, N$)
    $o_s \leftarrow$ Observation($s, \mathcal{R}_{\text{upcoming}}$)
    $P_A(s, \rho) \leftarrow \pi_s(o_s)$ **for** $\rho \in \mathcal{R}_{\text{upcoming}}$
  end for
  $A \leftarrow$ SelectActions(P_A)
  return A
end function

Options for the three functions that ComputeJointAction is parameterized by—NeedRetasking,

Algorithm 6 Update request lists by communicating

```
function UpdateRequestSets(𝕊, 𝕊_R, 𝒰, ℱ, ℐ)
    for s ∈ 𝕊 do
        if TaskSuccessful(s) then
            𝒰 ← 𝒰 \ {Task(s)}
            ℱ ← ℱ ∪ {Task(s)}
        end if
    end for
    ℐ ← ∅
    for s ∈ 𝕊 \ 𝕊_R do
        ℐ ← ℐ ∪ {Task(s)}
    end for
    return 𝒰, ℱ, ℐ
end function
```

AvailableRequests, and SelectActions—are described below.

6.3.2 Retasking Criteria

Two options are considered for use as the NeedRetasking function in Algorithm 5: RetaskAll and RetaskNecessary.

(1) The first retasking criterion, RetaskAll (Algorithm 7), trivially causes the policy to be reevaluated for all satellites in the constellation every time any satellite completes an action. This is motivated by the fact that the environment changes when a satellite completes a task, and as a result, the policy may react differently to this changed environment. It is possible to continue the previously selected task by reselecting it.

Algorithm 7 Retask all satellites

```
function RetaskAll(𝕊)
    return 𝕊
end function
```

(2) The second retasking criterion, RetaskNecessary (Algorithm 8), limits retasking to only when a satellite no longer has a task to carry out (due to one of the conditions described in the Transition Function being met). While actions may grow "stale" while using this method, it forces the policy to remain committed to a choice, while RetaskAll can lead

to "chatter" between actions.

Algorithm 8 Retask only necessary satellites

```
function RetaskNecessary(𝕊)
    𝕊_R ← ∅
    for s ∈ 𝕊 do
        if TaskComplete(σ, Task(s)) for any σ ∈ 𝕊 or TimedOut(s) then
            𝕊_R ← 𝕊_R ∪ {s}
        end if
    end for
    return 𝕊_R
end function
```

6.3.3 Deconfliction Methods

The implementations of AvailableRequests and SelectActions are closely intertwined. First, a completion-based approach to the functions is considered that reduces communication and computation requirements at the expense of optimal task deconfliction. Second, an intent-based approach that guarantees perfect task deconfliction is presented.

The completion-based set of methods (UnfulfilledRequests for AvailableRequests and GreedySelect for SelectActions, in Algorithm 9) requires communication only to update the fulfilled set; the intent set is not used. Satellites select between all unfulfilled requests and are tasked with the option most desired by the policy. This can be carried out entirely locally and in parallel on each satellite, but has the downside of having no mechanism to prevent multiple agents from selecting the same request at the same time.

Algorithm 9 Completion-based deconfliction

```
function UnfulfilledRequests(𝒰, ℐ)
    return 𝒰
end function
function GreedySelect(P_A)
    A[s] ← arg max_ρ(P_A(s, ρ)) for s ∈ 𝕊_R
    return A
end function
```

The intent-based set of methods (UnclaimedRequests for AvailableRequests and Hun-

garianSelect for SelectActions, in Algorithm 10) have a higher communication overhead but prevent task conflicts. In this method, agents are not able to select requests from the intent set. Once the request selection probabilities are computed by the satellite, an optimization problem in the form of the Hungarian problem is formulated and solved [153]: assign one request to each satellite such that no request is duplicated and the sum of selected request probabilities[1] is maximized across all satellites:

$$\begin{aligned} \text{maximize} \quad & \begin{bmatrix} P_A(\mathbb{S}_{R,1}, \rho_1) & \dots & P_A(\mathbb{S}_{R,1}, \rho_i) \\ \vdots & \ddots & \vdots \\ P_A(\mathbb{S}_{R,s}, \rho_1) & \dots & P_A(\mathbb{S}_{R,s}, \rho_i) \end{bmatrix} \mathbb{A}^T_{|\mathbb{S}_R| \times |\mathcal{R}|} \\ \text{such that} \quad & \mathbb{A}_{si} \in \{0, 1\} \\ & \sum_i \mathbb{A}_{si} = 1 \ \forall \ s \in \mathbb{S}_R \\ & \sum_s \mathbb{A}_{si} \leq 1 \ \forall \ i \in \mathcal{R} \end{aligned} \tag{6.2}$$

where $\mathbb{A}$ is a one-hot encoding of the action corresponding to each request for each satellite. Since $P(s, \rho) = 0$ for $\rho \notin \mathcal{R}_{\text{upcoming}}$, the weight matrix is sparse and any columns with zero weights can be removed from both that matrix and $\mathbb{A}$. To solve this problem, the probabilities must all be sent to an arbitrarily-selected chief satellite that completes the computation and returns action assignments. When only a single satellite is retasking ($|\mathbb{S}_R| = 1$) on a given step, HungarianSelect reduces to GreedySelect.

Because HungarianSelect can require some centralized computation and a high transmit/receive load on a single satellite, SequentialSelect is offered as a more distributed alternative. In this method, an arbitrary and random sequence is assigned to satellites that require retasking. One-by-one, each selects a task greedily then updates the intent set accordingly. As

[1] The sum of probabilities is used as a heuristic for the most desirable solution; if one was using a learning algorithm that estimated the state-action value function, such as SAC, the sum of values would be a more mathematically justified quantity to maximize.

Algorithm 10 Intent-based deconfliction

function UNCLAIMEDREQUESTS($\mathcal{U}, \mathcal{I}$)
    return $\mathcal{U} \setminus \mathcal{I}$
end function
function HUNGARIANSELECT(P_A)
    Construct Equation 6.2
    $\mathbb{A} \leftarrow$ solve Equation 6.2
    $A \leftarrow$ PARSEONEHOT($\mathbb{A}$)
    return A
end function

a result, task deduplication is still ensured. In implementation, the computation of UNCLAIMEDREQUESTS, UPCOMINGREQUESTS, OBSERVE, and π_s in COMPUTEJOINTACTION are unused and can be skipped.

6.3.4 Evaluated Combinations

Five configurations of the algorithm are explored in this work, enumerated in Table 6.1. Cases identified by **C-*** use methods that only consider the list of **C**ompleted requests when tasking; **IH-*** and **IS-*** indicate that **I**ntents are also shared and utilized, using the **H**ungarian and **S**equential methods respectively. The retasking frequency is considered for retasking **A**ll satellites at each step (***-A**) or only **N**ecessary satellites (***-N**). Additionally, the **N**o **C**ommunication case (**N.C.**) is used as a baseline for non-collaborative tasking. Each satellite is treated independently, maintaining its own list of fulfilled requests that is not updated by others in the constellation. Finally, the previously described MILP solution provides an open-loop solution and an upper bound on the optimal solution.

6.4 Algorithm Comparison

The algorithms defined in Table 6.1 are used with policies trained in a single-agent environment from chapter 5 in four-satellite constellations with varying true anomaly spacing. Each algorithm's performance is tested over a range of request densities and distributions (Figure 3.3). First, the behaviors of the methods are compared to each other, considering metrics including the

Algorithm 11 Lower compute and communication alternative to HungarianSelect

function SequentialSelect(P_A)
  $A \leftarrow \emptyset$
  for $s \in \mathbb{S}_R$ **do**
    $\mathcal{R}_{\text{avail}} \leftarrow$ UnclaimedRequests($\mathcal{U}, \mathcal{I}$)
    $\mathcal{R}_{\text{upcoming}} \leftarrow$ UpcomingRequests($s, \mathcal{R}_{\text{avail}}$)
    $o_s \leftarrow$ Observe($s, \mathcal{R}_{\text{upcoming}}$)
    $P_A(s, \rho) \leftarrow \pi_s(o_s)$ **for** $\rho \in \mathcal{R}_{\text{upcoming}}$
    $A[s] \leftarrow \arg\max_\rho(P_A(s, \rho))$
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{A[s]\}$
  end for
  return A
end function

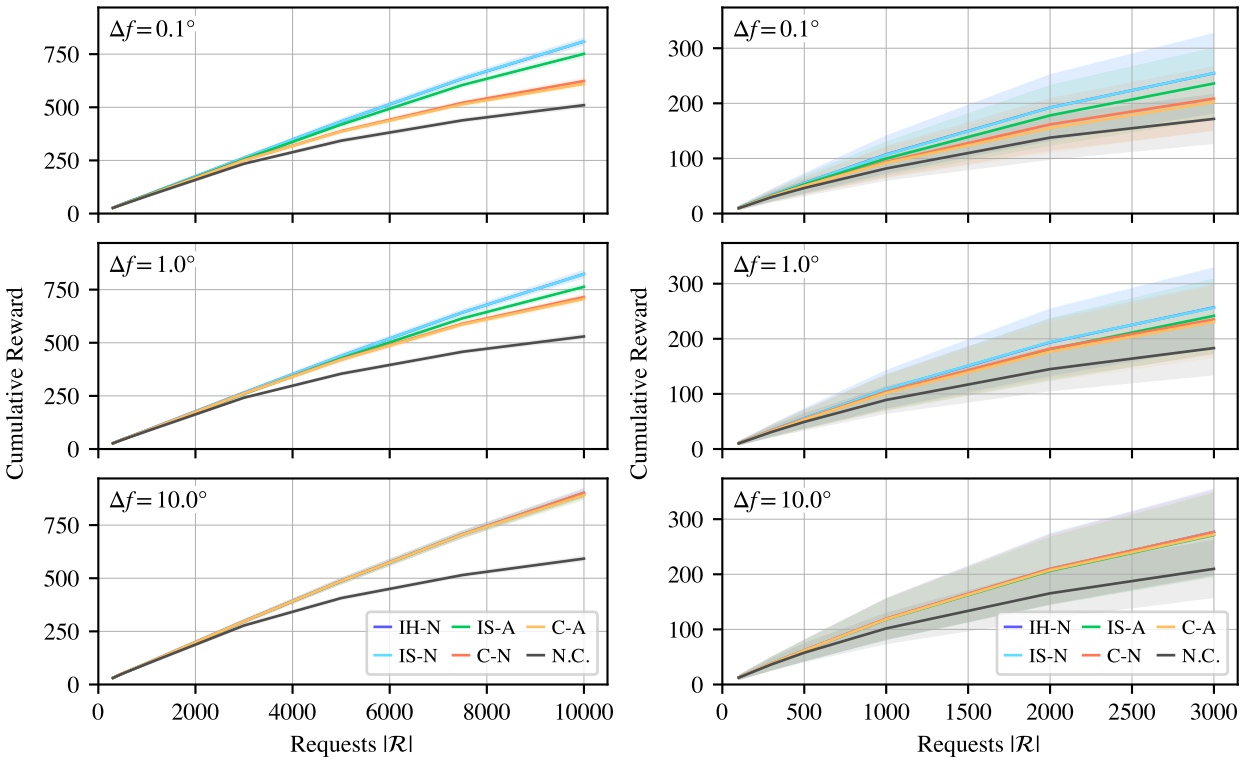
overall reward, number of duplicated requests, and amount of wasted time. Then, the coordination methods are compared to the bounds on optimal solution given by the multi-agent MILP solver from chapter 4. To demonstrate the scalability of the methods, the computational cost and performance of the **IS-N** method is compared to the MILP solver on increasingly large problem instances.[2]

6.4.1 Algorithm Performance

Each communication algorithm is evaluated in 20 randomly-seeded environments for a three orbit planning horizon at each density and distribution of requests. The performance is evaluated on the cumulative reward obtained by the four satellites, the count of unique and duplicated requests fulfilled, and the amount of time wasted. Wasted time is defined as time that a satellite spends towards fulfilling a request but aborts fulfillment before completion due to another satellite fulfilling the same request first. It is reported as a fraction of the total satellite·hours in the simulation.

Figure 6.1a and 6.1b show the cumulative rewards for each constellation spacing over uniform and city requests, respectively. In all cases, some communication strategy is significantly better than no communication. **IH-N** and **IS-N** perform best, followed by **IS-A**, **C-N**, and **C-A**. The difference in algorithm performance is greatest when satellites are close together and requests are dense, but with larger satellite spacing, request deconfliction is less urgent; therefore, the specific

[2] Visualizations of select cases are available at https://www.youtube.com/watch?v=lCN0TiNJ1i4.



(a) Uniform request distribution.

(b) City request distribution.

Figure 6.1: Comparison of algorithm performance on 4-satellite "String" constellation with varying Δf.

method used to deconflict tasks is less important. The only-necessary retasking methods (**-N**) perform better than their corresponding always-retasking methods (**-A**); this implies that the policy is "unfocused" and switches tasks when given repeated opportunities to retask, as opposed to when the policy is forced to stay with the task it initially selected. This is not surprising, since frequent mid-task retasking is outside the policy's training domain.

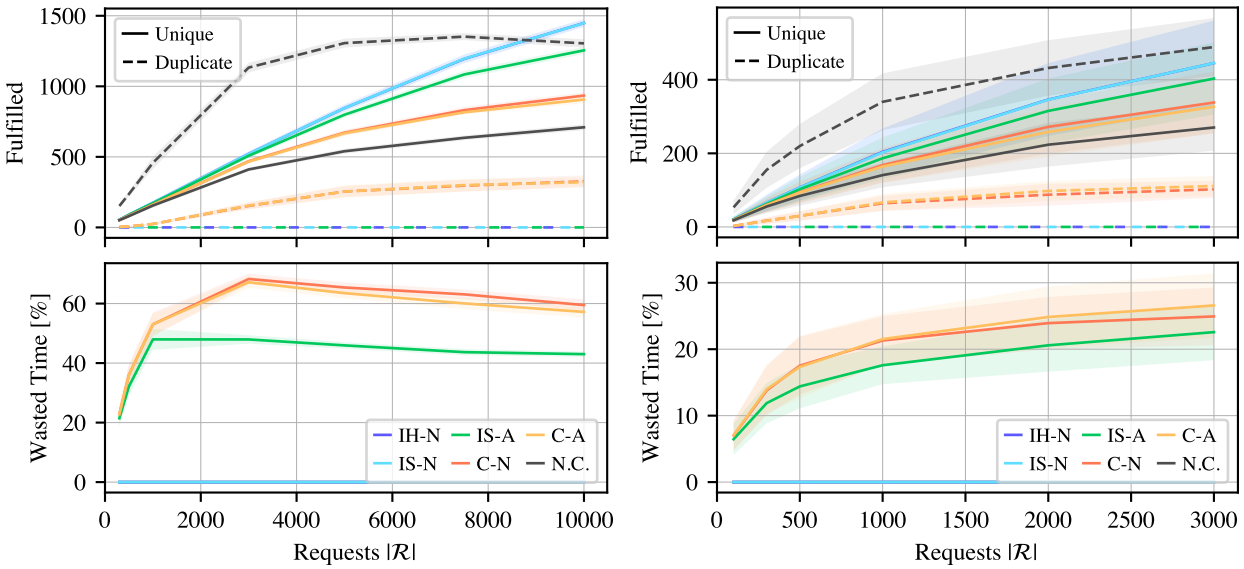
Request fulfillment counts and wasted time are given in Figure 6.2a for uniform requests and Figure 6.2b for city requests. With no deduplication mechanism, the no-communication case exhibits a very high request duplication count, with requests generally being imaged three or more times. The completion-based methods also show some request duplication, which can happen if two satellites with the same task both complete the task at about the same time; with larger inter-satellite spacing, this becomes increasingly rare. As the algorithm designs would suggest, the intent-based methods prevent any duplication of images. Nonzero wasted time is only observed for **IS-A**, **C-N**, and **C-A**, since **IH-N** and **IS-N** never allow satellites to attempt a task that another satellite is already attempting; the metric is not considered for the no-communication case because it is not well-defined without communication.

Between the cumulative reward, fulfillment counts, and wasted time metrics, the Hungarian (**IH-N**) and sequential (**IS-N**) methods exhibit equivalent performance. While the Hungarian method is more theoretically justified at a higher computation cost, the results imply that same-step retasking of multiple satellites in which both satellites prefer the same task is rare enough that the method used to resolve the problem is immaterial.

Another consideration for constellation performance is the per-satellite load; i.e., is one satellite completing more tasks and thus collecting more data than others? Preferably, all satellites should be responsible for an equal number of completed tasks. Figure 6.3 shows the relative rewards obtained by each satellite in the four-satellite lead-follower configuration. **IH-N** and **IS-N**, which are already the best by other metrics, are seen to distribute task load equally across the constellation. The other methods all unfairly result in the lead satellite being responsible for more tasks than the followers.

Table 6.1: Evaluated collaboration algorithm configurations.

Case	UnfulfilledRequests	UnclaimedRequests	GreedySelect	HungarianSelect	SequentialSelect	RetaskAll	RetaskNecessary
C-A	✓		✓			✓	
C-N	✓		✓				✓
IS-A		✓			✓	✓	
IS-N		✓			✓		✓
IH-N		✓		✓			✓
N.C.	No communication						
MILP	Open-loop planner from [46]						



(a) Uniform request distribution.

(b) City request distribution.

Figure 6.2: Statistics for the $\Delta f = 0.1°$ string constellation.

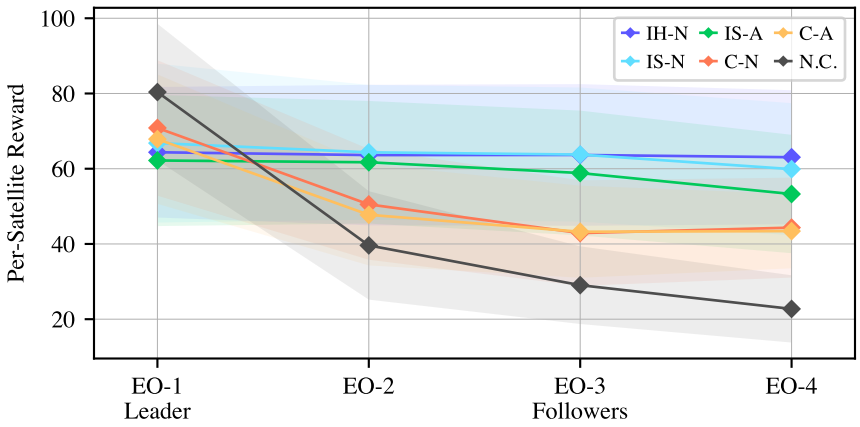
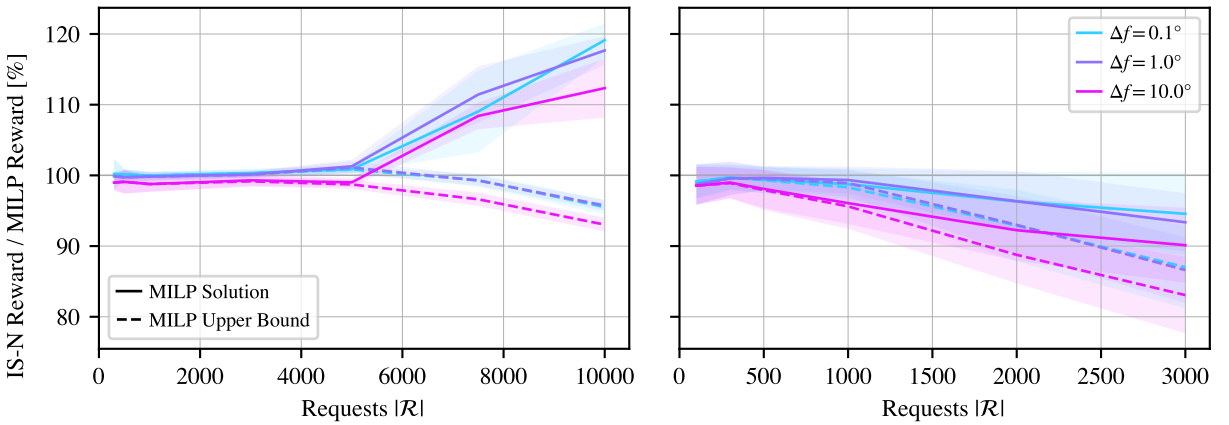


Figure 6.3: Per-satellite rewards; city requests, $|\mathcal{R}| = 3000$, $\Delta f = 0.1°$.

6.4.2 Optimality Comparison

To quantify the performance of the best method, **IS-N**, the results are compared to MILP solutions using the solver from chapter 4 on the same environments. The MILP solver is given two hours of solve time to return a suboptimal solution (which is also a lower bound on optimality) and an upper bound on optimality; if the two are equal, the solution is provably optimal. MILP solutions are executed in the same simulation environment as **IS-N** is evaluated in to demonstrate feasibility.

Figure 6.4 shows a comparison of the performance of **IS-N** versus optimality bounds from the MILP solution across request distributions. Across uniform requests, **IS-N** produces results that are up to 18% better than the MILP solution, and at worst 7% below the upper bound. With larger (i.e., harder) problems, **IS-N** more greatly outperforms the MILP solution, which struggles even with considerable compute time. As with the optimality comparison in chapter 5, the city request distribution is more challenging for the policy. Still, with low-cost, closed-loop planning, **IS-N** is able to obtain rewards within 10% of the MILP solution and within 18% of the upper bound.



(a) Uniform request distribution.

(b) City request distribution.

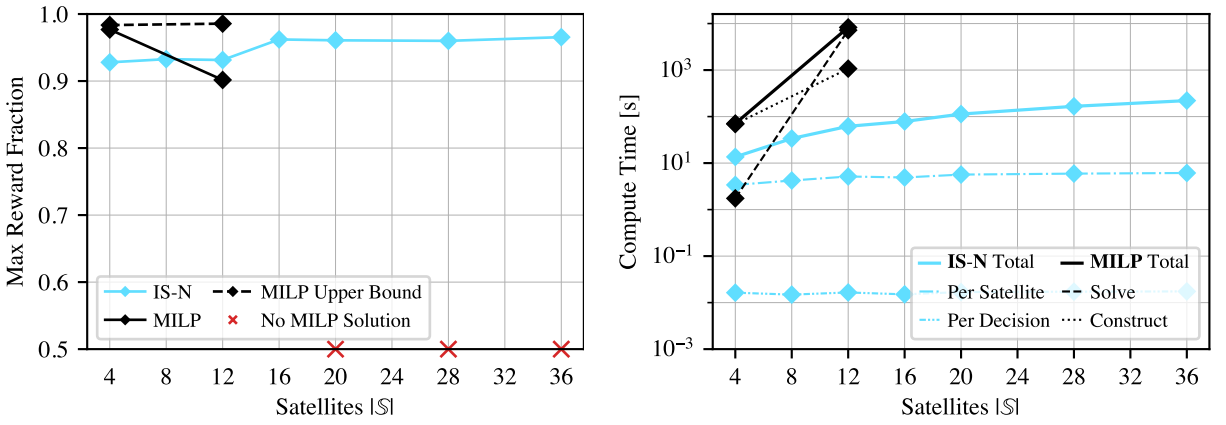
Figure 6.4: Comparison of IS-N to MILP optimality bounds, $N = 4$ satellite "String" constellation.

6.4.3 Scalability Demonstration

Finally, the scalability and closed-loop behavior of the RL-based approach is explored on larger constellations with more requests. A Walker-Delta constellation is constructed with $|\mathbb{S}|$ satellites evenly distributed across four 45° × 800 km planes, with up to 9 satellites per plane for a total of 36 satellites [135]. Instead of a static set of requests, the environment starts with $|\mathcal{R}|_0 = 100 \cdot |\mathbb{S}|$ requests and has new requests appear at a rate of $|\dot{\mathcal{R}}| = 100 \cdot |\mathbb{S}|$ requests per hour over the three-orbit planning horizon; the largest environment features 20,700 requests.

The **IS-N** method is compared against the MILP solver in this environment. **IS-N** operates in a closed-loop fashion, only considering new requests as they appear in the environment. The MILP solver is given the advantage of perfect *a priori* knowledge of request appearances to avoid having to consider periodic replanning. As in the previous MILP cases, the solver is given unlimited setup time and up to two hours of solve time to return a solution. As an absolute upper bound on performance not considering feasibility, the maximum reward available in the environment is also computed as the sum of the priorities of all requests that have at least one imaging opportunity.

The scalability benefits of the distributed, RL-based approach over the MILP solution are clear in Figure 6.5, even with the knowledge advantage given to the MILP solver. The RL policy



(a) Total reward comparison.

(b) Compute time comparison.

Figure 6.5: Comparison of IS-N and MILP performance and computation time across varying Walker-Delta constellation sizes.

consistently achieves about 94.9% of the total reward available in the environment regardless of constellation size; comparatively, the MILP solver only has strong performance for the smallest constellations, and performance drops off quickly with the solver unable to converge on any solution for $|\mathbb{S}| \geq 20$ satellites within two hours. Furthermore, in a realistic scenario with newly appearing requests they would not be known to the MILP solver before they appear; in practice, this would lead to—at minimum—a delay equal to the solution time between appearance and fulfillment. The distributed computational cost of the RL policy is very low, averaging 15 ms per decision. With these experiments, the method's consistent performance across constellation scales and when encountering new requests is demonstrated.

Chapter 7

Distributed, Autonomous Tip-and-Cue Operations

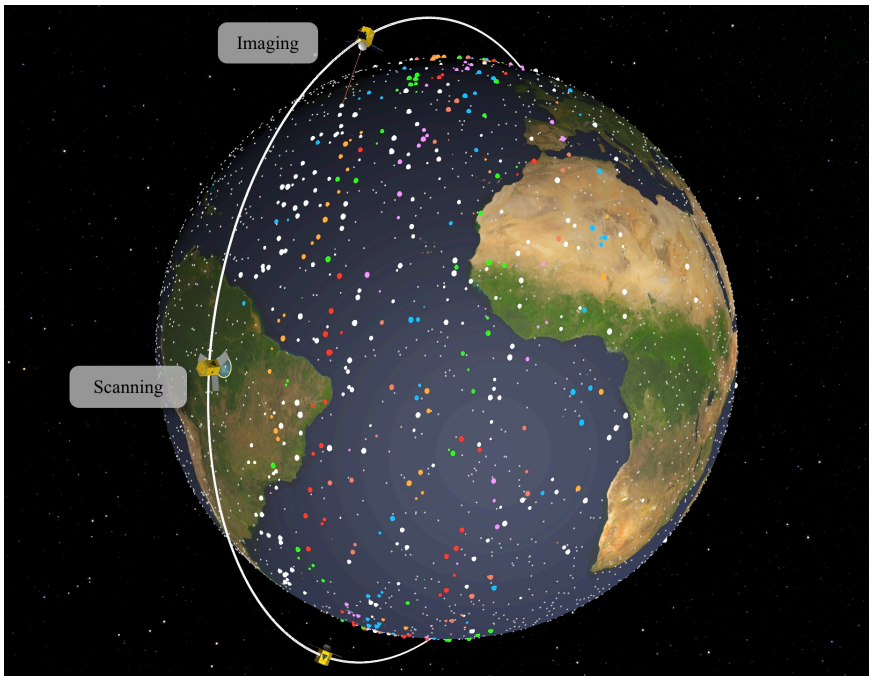


Figure 7.1: Visualization of LEO satellites in the scanning and imaging modes with unknown (gray), found (white), and imaged (colored) targets, configured in the $S = 6$ satellite Ring constellation.

7.1 Motivation

While Earth-observing constellations have become increasingly prevalent, the operation of these systems has largely developed within a self-limiting framework: Given a list of *a priori* requests of varying values, operators create a global schedule that maximizes the output of the constellation subject to operational constraints [6]. However, the increasing availability of increasingly

capable satellites (both with respect to sensing ability and onboard compute power) enables previously unavailable mission architectures [2]. In particular, emergent phenomena—such as natural disasters, human events, and scientifically interesting occurrences—cannot be responsively imaged in the existing paradigm, but distributed autonomy would allow constellations to adapt in order to exploit unpredictable and unanticipated events [3]. Such a system would extend existing tip-and-cue behavior, in which a leader "tips" following satellites about events of interest, to an automated, distributed, constellation-wide tip-and-cue architecture [154].

Decentralized job allocation in constellations has been considered previously in order to avoid the computational expense of global planners. An early deep reinforcement learning (DRL) algorithm is applied in Reference 155 to combine individual and collaborative task planning for satellites. In Reference 156, self-adaptive complex system theory is used for multi-satellite mission planning, with a focus on flexibility and robustness. References 157 and 158 formulate the allocation problem as a distributed constraint optimization problem (DCOP), allowing a variety of algorithms with different levels of communication requirement to be applied. Unlike this chapter, this prior work and the previous chapters all still assume an *a priori* task list to be distributed to satellites.

The necessity of having responsive satellites has also been well-established in the literature. One of the earliest examples of using data collected onboard to update the mission plan in a closed-loop manner was on Earth Observing-1 (EO-1) [159]. Over time, EO-1 and other assets have been used as part of a larger sensor web in order to automate detection, tasking, and data acquisition [160]. More recently, autonomous responsive tasking has been demonstrated and flown in a single-satellite architecture in a method known as dynamic targeting [16, 161, 17]. This system uses a forward-looking sensor to identify obstructed or valuable regions of the upcoming ground track, then points the imaging instrument accordingly. Other work studies decentralized consensus-based algorithms for reactive replanning of Earth-observing satellites (EOSs), finding that the efficacy of replanning for reobservation of new events is highly dependent on constellation geometry [149, 150].

This chapter expands the Markov decision process (MDP) formulation of the standard agile Earth-observing satellite scheduling problem (AEOSSP) given in chapter 3 into a tip-and-cue

problem with a constellation of agile low Earth orbit (LEO) satellites (Figure 7.1).[1] Instead of being given an *a priori* request list, satellites are equipped with a wide field of view (FOV) scanning instrument that identifies new imaging targets and a narrow FOV, wide field of regard (FOR) imaging instrument that can collect reward-yielding images of those targets. To perform this task well, agents must learn a decentralized policy that balances searching for new targets and exploiting known targets; ultimately, this must be a collaborative behavior between agents. In this chapter, policies are learned and tested across a variety of constellations. By examining the individual and collective behaviors of the satellites, it is apparent that agents learn to work together by diversifying roles within the constellation and increasing overall performance.

7.2 Problem Formulation

The search-and-image problem formulation (Figure 7.2) is a modification of the standard AEOSSP environment described in chapter 3. The primary changes are to the appearance of potential targets in the environment, how targets are identified by the satellites, and how information is shared within the constellation.

7.2.1 Modifications to the AEOSSP

The dynamics of collecting images and the instruments used to do so must be modified from the AEOSSP to create the tip-and-cue environment.

7.2.1.1 Target Model

In the problem, each target is defined by a tuple of Earth-fixed location, priority, and appearance time $\tau_i = (\boldsymbol{r}_i, r_i, t_i)$. When a new target appears at time t_i, it is added to the set of all targets $\mathcal{T}$. New targets appear in the environment at the rate $\dot{\tau}$, which is randomized between 100 and 1000 targets per hour, with locations uniformly distributed over Earth's surface. When a satellite identifies a previously unknown target using its scanning instrument, the target is added

[1] Visualizations of select cases are available at https://www.youtube.com/watch?v=4orleGCi7n0.

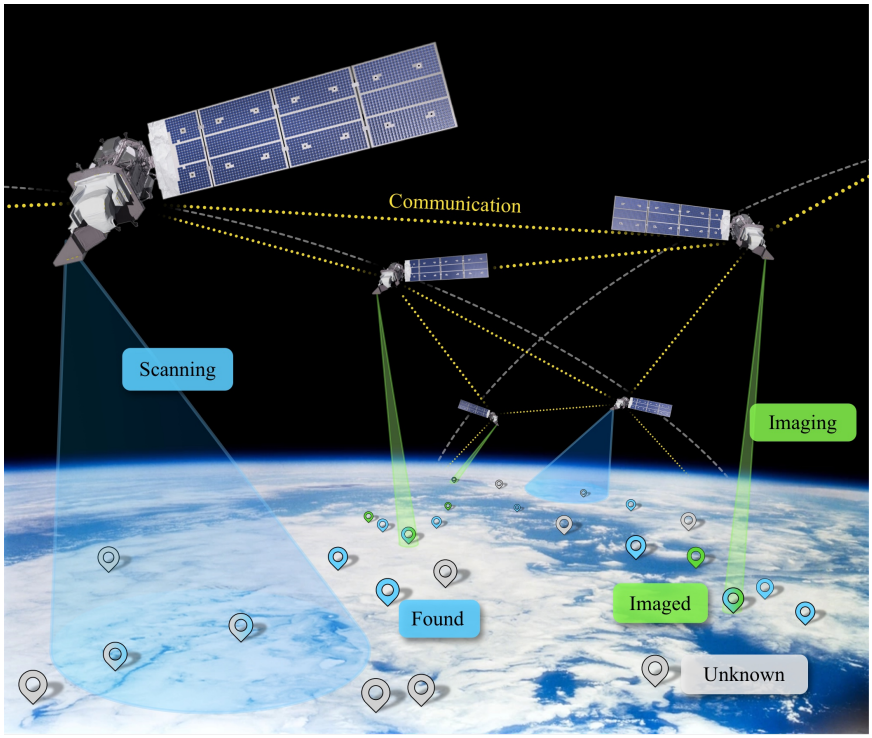


Figure 7.2: Constellation architecture for homogenous satellites equipped with scanning and imaging instruments.

to the set of known targets $\mathcal{K}$. Once a target is known, it may be imaged with a satellite's imaging instrument; this adds the target to the set of imaged targets $\mathcal{I}$ and yields a reward equal to the target's priority $r_i \in [0, 1]$. Each of these sets is a subset of the previous: $\mathcal{I} \subset \mathcal{K} \subset \mathcal{T}$.

The objective remains the same as that in the AEOSSP: maximize the sum of priorities of imaged targets. However, the dynamics of target fulfillment must be satisfied to do so.

7.2.1.2 Instrument Models

Each satellite in the homogeneous constellation has two instruments: a scanning instrument for identifying unknown targets and an imaging instrument for collecting images of known targets. The imaging instrument works as described in the standard AEOSSP formulation. It is a camera with a body-fixed boresight $\hat{\boldsymbol{c}}$ that is aimed using the agilely maneuvering satellite; the image is automatically collected once the attitude controller [131] has settled. This narrow FOV instrument can collect images of targets with an elevation angle $\phi > 58°$, which corresponds to a circular FOR

with a 500 km radius. The scanning instrument is used when the satellite is in a nadir-pointing attitude to reveal unknown targets within the 500 km-radius FOV, equal to the FOR of the agile imaging instrument. When activated, the instrument follows a sequence consisting of 45 seconds of warm-up and slewing time, 90 seconds of active scanning time, and 45 seconds of image processing time, during which targets scanned with the instrument are identified.

7.2.1.3 Communication

Free communication is assumed within the constellation. Once every 50 seconds, satellites broadcast a list of newly scanned and imaged targets to update the constellation's knowledge of the environment. While this degree of constellation-wide communication is unrealistic for current systems, it is justified in two ways: advances in in-space communication networks reasonably imply that bandwidth may become a less restricted resource, and prior research has demonstrated that in similar scenarios, local-only and global communication produce similar results since only activities from other physically proximate satellites impact decision-making in a time-sensitive way [107].

7.2.1.4 Constellation Geometry

Two constellation geometries are considered in this chapter, both parameterized by the number of satellites S. The **Ring** constellation consists of S satellites equally spaced in a single-plane 9000 km orbit. The **String** constellation consists of S satellites separated by 10° true anomaly in a single-plane $60° \times 800$ km orbit.

7.2.2 MDP Formalization

In order to find a policy for this problem, it must be formulated as a decentralized partially-observable semi-Markov decision process (Dec-POsMDP). The elements of the MDP are defined as follows:

- **State Space:** The state space consists of all information about the environment and the dynamics simulation necessary to propagate the environment. Since the total state is very

Table 7.1: Observation vector elements; request observations are given for next $N = 32$ upcoming unimaged targets in $\mathcal{K} \setminus \mathcal{I}$.

Quantity	Dim.	Description
${}^{\mathcal{H}}\boldsymbol{\omega}_{\mathcal{BE}}$	3	Body angular rate
${}^{\mathcal{H}}\hat{\boldsymbol{c}}$	3	Hill-frame instrument direction
${}^{\mathcal{E}}\boldsymbol{r}_{\mathcal{BE}}$	3	Earth-fixed position
${}^{\mathcal{E}}\boldsymbol{v}_{\mathcal{BE}}$	3	Earth-fixed velocity
$r_{n\in N}$	N	Target priorities
${}^{\mathcal{H}}\boldsymbol{r}_{n\in N}$	$3N$	Target positions
$\delta\theta_{n\in N}$	N	Target pointing errors
$t^o_{n\in N}, t^c_{n\in N}$	$2N$	Target opportunity windows

high dimensional and most of these states are irrelevant to decision-making, a hand-designed observation space is selected.

- **Observation Space:** The per-satellite observation is as given in Table 7.1. The observation consists of relevant information about the satellite, such as its position, velocity, attitude, and pointing direction, as well as information about the next $N = 32$ along-track targets in the known-but-not-imaged set $\mathcal{K} \setminus \mathcal{I}$. Target information includes Hill-frame $\mathcal{H}$ relative position and the angle $\delta\theta$ between the imaging instrument boresight $\hat{\boldsymbol{c}}$ and the target. The positions of the upcoming targets provide insight into whether the satellite should scan to find more targets or try to image known targets.

- **Action Space:** Each satellite has $N + 1 = 33$ actions available. Imaging actions $a_{\text{im},i}$ account for 32 of the actions. With this action, the satellite attempts to image the corresponding target in the observation space. The action is not guaranteed to be successful, as the target may go out of range before the controller has settled to point the instrument at the target. If the action is successful, the target is added to the imaged set $\mathcal{I}$. The satellite also has the scanning action a_{scan}. This action activates the previously described sequence

of warm-up, scan, and cool-down, then adds any detected targets to the known set $\mathcal{K}$.

- **Reward Function:** The reward function yields rewards equal to the priority of imaged targets. With $\mathcal{I}_s$ being a satellite's imaged set before the step and $\mathcal{I}'_s$ after the set, the reward at a step for satellite s is

$$r_s(\mathcal{I}_s, \mathcal{I}'_s) = \sum_{\tau_i \in \mathcal{I}'_s \setminus \mathcal{I}_s} r_i \tag{7.1}$$

  unless multiple satellites have imaged the same target at the same step, in which case the reward is distributed evenly among them.

- **Transition Model:** Transitions are given by a generative model (i.e., a simulator). At each step, the simulation is propagated until any satellite completes an action. When that action is done, a new action is selected for that satellite and the simulation continues. Since actions can take different amounts of time, this results in the satellites acting asynchronously. Episodes are propagated for 15 orbits before the environment is reset and rerandomized.

As with environments in other chapters, the MDP is implemented using BSK-RL.

7.2.3 Training Pipeline

Proximal policy optimization (PPO) is used for training policies in a cooperative multi-agent reinforcement learning (MARL) context [34]. PPO is a widely-used DRL algorithm that has been demonstrated to perform well across domains such as games and robotics. It has previously been demonstrated on other Earth-observation scheduling problems [162]. The RLlib implementation of PPO is used [129], modified to handle asynchronicity and semi-Markov intervals.[2]

As shown in Figure 7.3, the MARL training pipeline uses the centralized training, decentralized execution paradigm. Each satellite generates separate experience tuples and independently determines actions, but all satellites use copies of the single policy and contribute their experience

[2] Examples of these modifications are given in https://avslab.github.io/bsk_rl/examples/time_discounted_gae.html and https://avslab.github.io/bsk_rl/examples/async_multiagent_training.html.

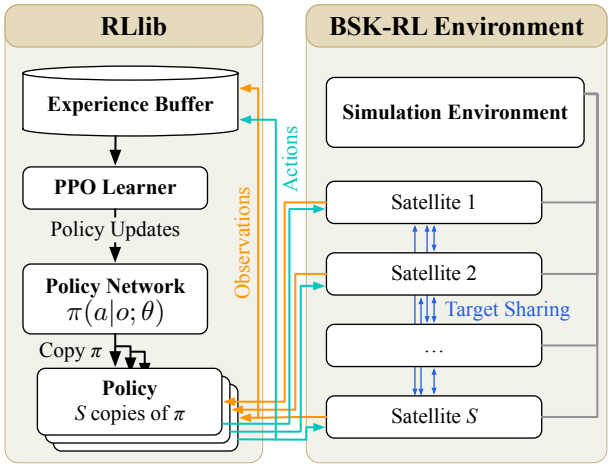


Figure 7.3: Schematic of MARL training.

toward training that single policy. As a result, the policy is not explicitly tied to a certain training constellation and can be deployed with any number of other satellites in the environment.

A hyperparameter search was performed to find a strong training configuration, yielding a learning rate of 3×10^{-5}, discount rate of 0.995 (with time units of s), and batch size of 100 samples per thread. All other parameters use the RLlib defaults. Policies are represented by a 2×2048 node multilayer perceptron (MLP). Each policy is trained for 48 hours on 32 threads; this corresponds to 3.3M to 3.7M environment interactions and 8.8 to 13.8 satellite-years on-orbit. Four policies are trained: String-3 and String-6 are trained on a String constellation with $S = 3$ and 6 respectively, and Ring-6 and Ring-12 are likewise trained on a Ring constellation. In each episode, the target appearance rate $\dot{\tau}$ is randomly sampled between 100 and 1000 targets per hour. Each episode is propagated for 15 orbits of decision-making. A policy Ring-1 is also trained to demonstrate the performance of a policy that learned in a single-agent environment.

7.3 Constellation Performance

To evaluate the policies, each is benchmarked across different constellations and target rates for 15 orbits; these are compared to a heuristic policy benchmark. The Ring and String constellations are both tested for S between 1 and 36 satellites; other than a difference in inclination (90°

versus 60°), the two constellations are the same for $S = 36$. The target rate $\dot{\tau}$ is varied between 100 and 1000 targets per hour, as the policy was exposed to during training. The size of the policy network makes its practical impact on storage and compute requirements negligible, with inference taking 10 to 20 ms at each decision.

7.3.1 Heuristic Policy

A heuristic policy is developed as a baseline for comparison. The policy must balance scanning and picking feasible imaging targets. The policy, parameterized by K, is given in Algorithm 12. In short, the heuristic tries to complete K images per the $t_{\text{scan}} = 180$ second scanning-action duration. If there are fewer than K upcoming targets, the satellite will scan; otherwise, it will attempt to image the upcoming target with the greatest priority per time until opportunity close among targets with greater than t_{scan}/K seconds until opportunity close (as a guess of what targets can be feasibly slewed to).

Algorithm 12 Heuristic Policy

function HEURISTIC-$K(o, K)$
  if $t^c_K > t_{\text{scan}}$ **then**
    return a_{scan}
  else
    $n^* \leftarrow \arg\max_n \left(r_n/t^c_n \mid t^c_n > t_{\text{scan}}/K\right)$
    return a_{im,n^*}
  end if
end function

7.3.2 Policy Benchmarks

Key metrics are examined to understand the behavior and performance of the policies under different conditions. The known percentage $\mathcal{K}_{\%}$ and imaged percentage $\mathcal{I}_{\%}$ reflect the fraction of targets with at least one opportunity that are respectively scanned and imaged by the constellation; this metric is used so that the performance is not diluted by targets that no satellite has access to over the course of the simulation purely due to geometry. Imaged per known $\mathcal{I}/\mathcal{K}$ gives the fraction of scanned targets that are eventually imaged. The scanning mode percent is the fraction of time

spent taking the scanning action (as opposed to an imaging action) on average by all satellites in the constellation. The total reward ΣR is the sum of the priorities of imaged targets across all satellites, i.e., the undiscounted sum of rewards. The normalized reward $\Sigma R/S\dot{\tau}$ is the previous metric normalized by the target appearance rate and the number of satellites.

Table 7.2: Total reward ($\mu \pm \sigma$) relative to Ring-6 policy across all benchmarks [%].

	Benchmark Constellation		
Policy	**Ring**	**String**	**All**
Ring-6	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
Ring-12	92.1 ± 9.7	86.3 ± 13.4	88.8 ± 12.3
String-3	91.1 ± 7.7	94.0 ± 6.7	92.7 ± 7.3
String-6	93.5 ± 8.1	97.8 ± 6.0	95.9 ± 7.3
Ring-1	84.2 ± 15.5	80.5 ± 16.4	82.1 ± 16.1
Heuristic-3	87.2 ± 11.3	88.2 ± 10.4	87.8 ± 10.8
Heuristic-6	81.1 ± 24.4	81.9 ± 26.5	81.6 ± 25.6
Heuristic-9	60.7 ± 30.3	59.8 ± 29.7	60.2 ± 30.0

Table 7.2 gives the average and standard deviation of per-seed relative performance of each policy against the Ring-6 trained policy. It shows that on average the Ring-6-trained policy is best, even if other policies slightly outperform it in certain cases. Even the heuristics occasionally outperform the learned policies, but they generalize poorly across all combinations of constellations and target appearance rates. The Ring-6 policy is used throughout the remainder of the chapter due to its good performance and strong ability to generalize.

The full benchmark of the Ring-6 policy on the Ring and String constellations is given in Figure 7.4 and Figure 7.5, respectively. One of the most important trends is in the $\mathcal{I}/\mathcal{K}$ ratio: as long as $S > 1$, at least a majority of scanned targets are imaged, with fractions $> 75\%$ in both constellations if $S \geq 6$. Unsurprisingly, increasing the number of satellites "saturates" the environment: with enough satellites, nearly all possible targets are scanned and imaged. As the number of satellites increases, the fraction of time spent in the scanning mode also increases in

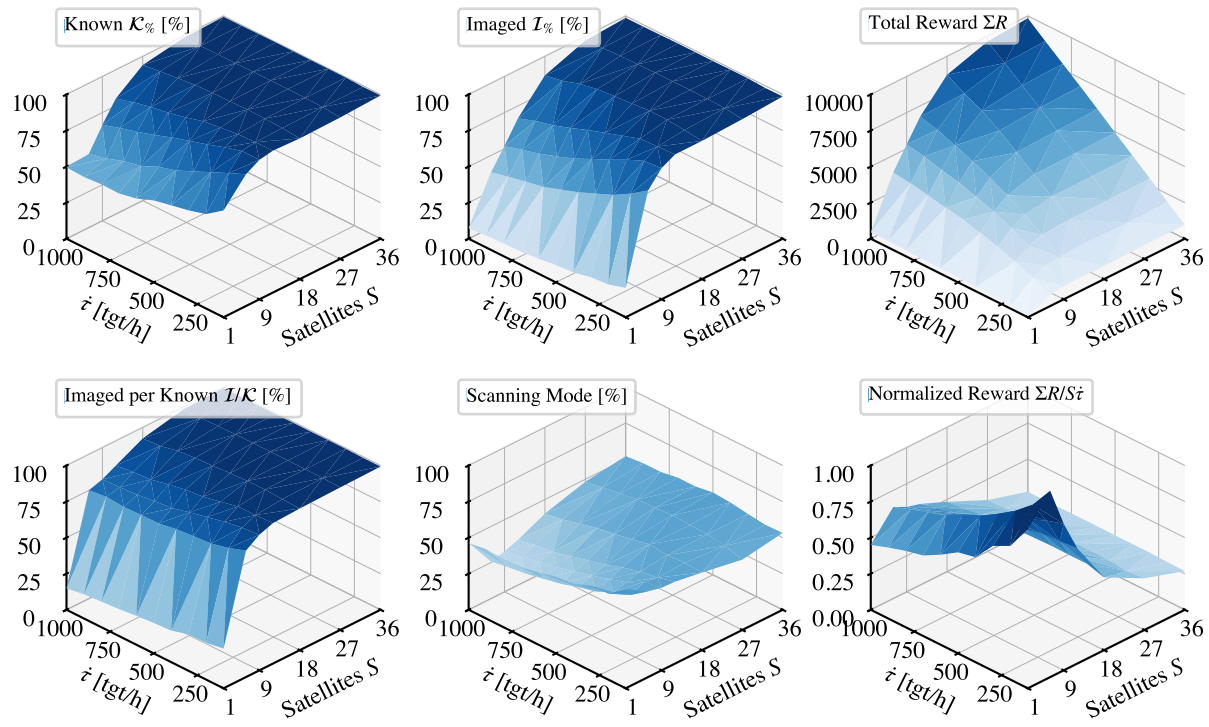


Figure 7.4: Ring constellation performance of the Ring-6 policy with varying target rate $\dot{\tau}$ and satellite count S.

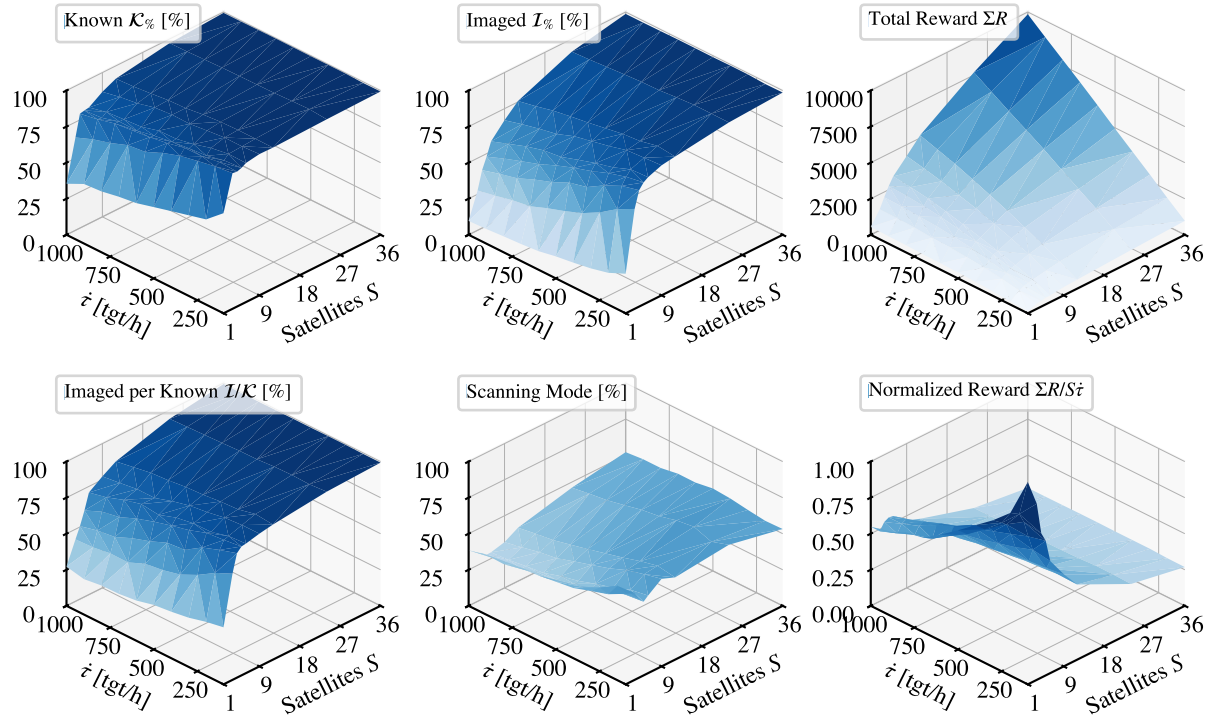


Figure 7.5: String constellation performance of the Ring-6 policy with varying target rate $\dot{\tau}$ and satellite count S.

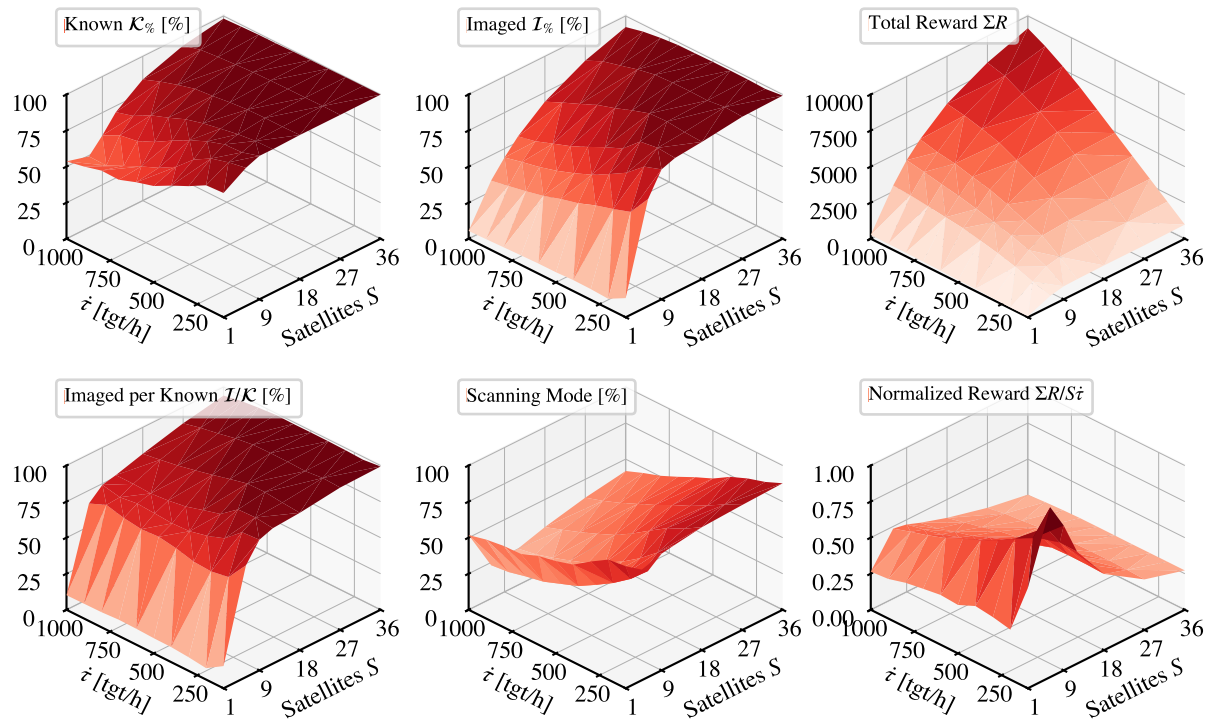


Figure 7.6: Ring constellation performance of the Ring-1 policy with varying target rate $\dot{\tau}$ and satellite count S.

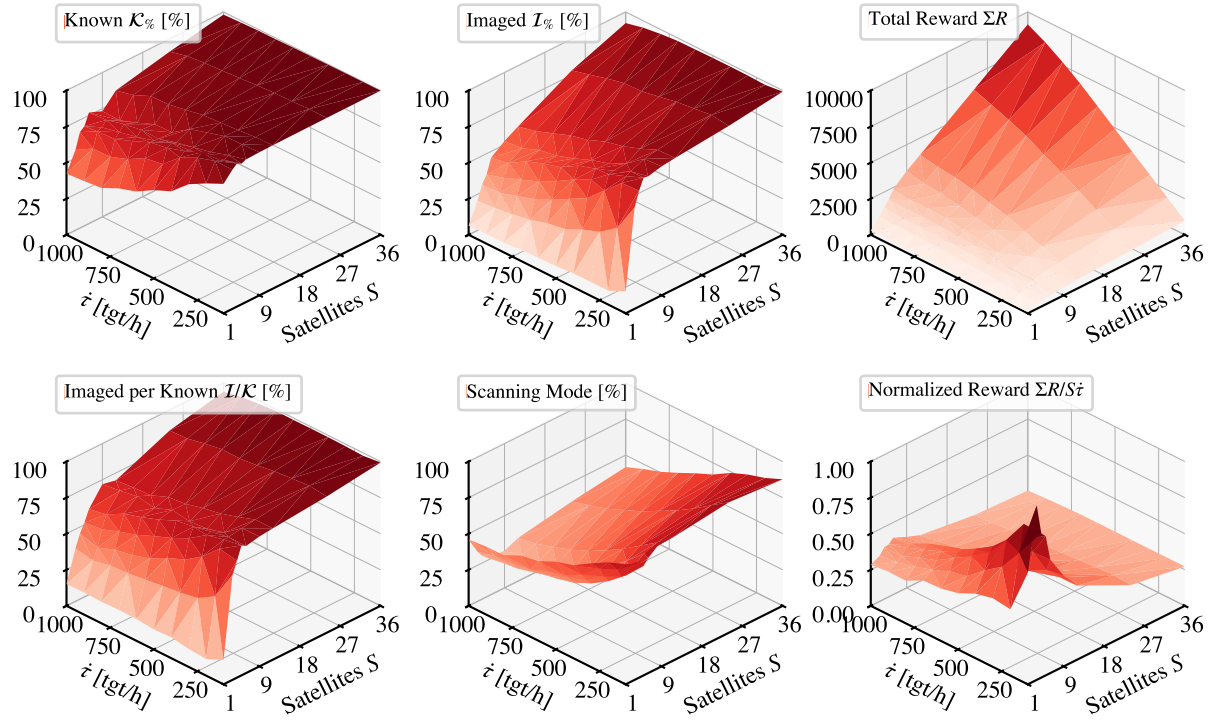


Figure 7.7: String constellation performance of the Ring-1 policy with varying target rate $\dot{\tau}$ and satellite count S.

order to avoid exhausting the known target list.

7.4 Collaboration Between Agents

While the overall constellation performance is strong, it is of interest to investigate how and where agents have learned to collaborate. Three hypotheses provide insight into this:

(1) If beneficial collaboration is occurring, performance should synergistically scale faster than the number of satellites in the constellation.

(2) If beneficial collaboration is learned in the multi-agent environment, it should yield better performance in multi-agent settings than a policy trained in a single-agent environment.

(3) Some agents may specialize with altruistic behaviors when collaborating.

These hypotheses can be confirmed in cases where beneficial collaboration is possible.

7.4.1 Per-Agent Improvement with Collaboration

One test for collaboration is to see whether increasing the number of satellites disproportionately improves performance. To do this, two metrics are analyzed: The per-satellite marginal increase in reward obtained for a constellation compared to a single agent

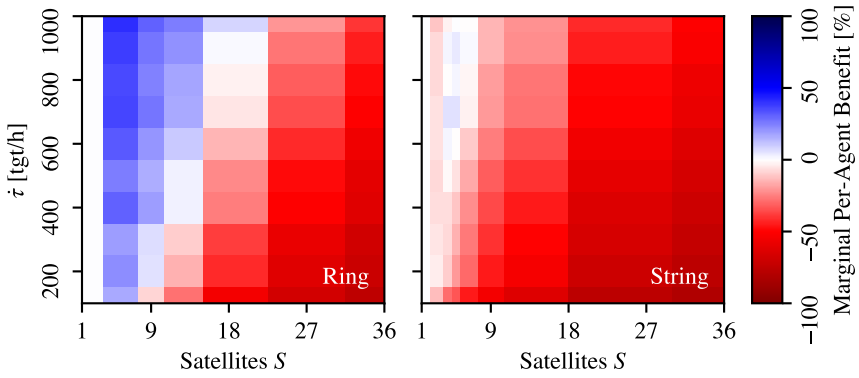
$$\frac{\text{ConstellationReward}(S, \dot{\tau})}{S \cdot \text{ConstellationReward}(1, \dot{\tau})} - 1 \tag{7.2}$$

and the same for number of images collected

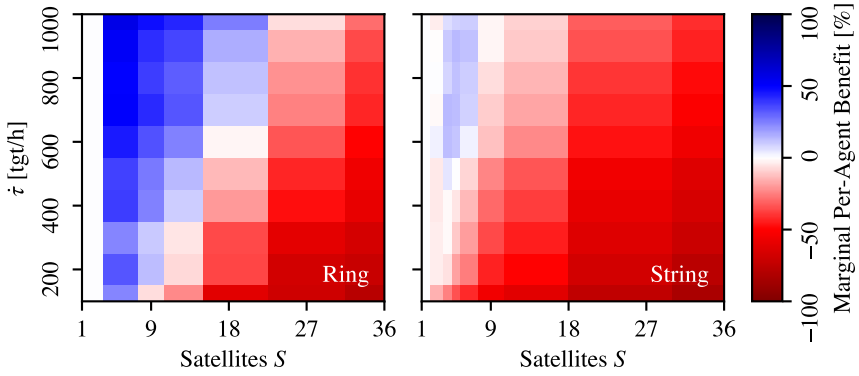
$$\frac{\text{ConstellationImages}(S, \dot{\tau})}{S \cdot \text{ConstellationImages}(1, \dot{\tau})} - 1 \tag{7.3}$$

where $\text{ConstellationReward}(S, \dot{\tau})$ and $\text{ConstellationImages}(S, \dot{\tau})$ are the respective metrics for an S satellite constellation at a target rate of $\dot{\tau}$.

Figure 7.8 plots the metrics for the Ring and String constellations. The red regions in the lower right corners of the heatmaps reflect cases where the constellation is undersubscribed: adding more satellites reduces per-satellite throughput because maximum target satisfaction for the



(a) Reward obtained (Equation 7.2).



(b) Targets imaged (Equation 7.3).

Figure 7.8: Per-agent marginal increase in performance of constellation versus single agent, Ring-6 policy.

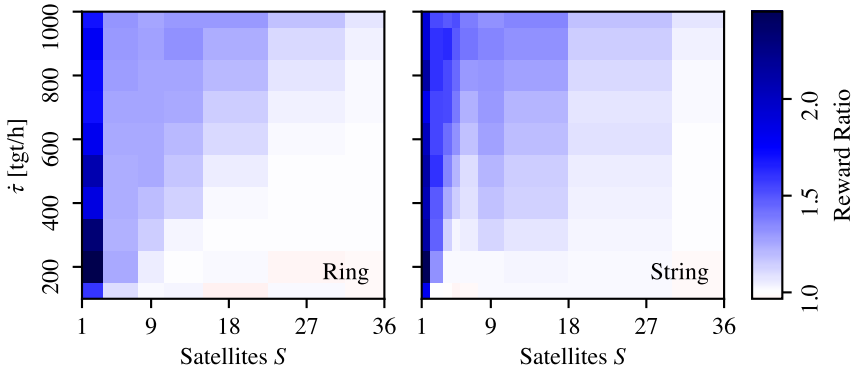


Figure 7.9: Ratio of rewards for multi-agent-trained policy Ring-6 to single-agent-trained policy Ring-1.

environment was already being achieved by a smaller number of satellites; colloquially, this is a "too many cooks in the kitchen" situation. However, the blue regions show cases where collaboration leads to stronger per-satellite performance. Here, adding more satellites improves constellation-wide performance at a rate greater than one.

Notably, the Ring constellation has a larger region of beneficial collaboration than the String constellation. This is a function of geometry, as adding more satellites to the String constellation does not greatly improve the variety of targets accessible to the constellation. Instead, followers far back in the String constellation pass over regions that have largely been depleted of targets by leading satellites. Similarly, the marginal improvement in reward is minimal for the String constellation because it quickly depletes high-value targets along the ground track.

7.4.2 Multi-Agent Learning

Another indicator of learned collaboration is that the multi-agent-trained policy Ring-6 outperforms the single-agent-trained policy Ring-1 across many cases. Figure 7.9 shows cases-by-case performance comparisons between the two policies, providing more detail than the aggregate comparison in Table 7.2. The greater the ratio is above 1, the more the multi-agent-trained policy has been able to leverage learned collaborative behaviors that the single-agent-trained policy has not learned. While the saturated cases (red regions in Figure 7.8) lack opportunities to exploit collaboration, cases where marginal per-satellite increases in performance are observed also exhibit increased performance from learning in a multi-agent environment.

7.4.3 Behavior Specialization

Collaboration is also implied by implicit role assignment or diversification of behaviors within the constellation. If a satellite acts differently due to the presence and actions of another satellite, and that positively impacts the constellation-wide performance, collaboration is occurring.

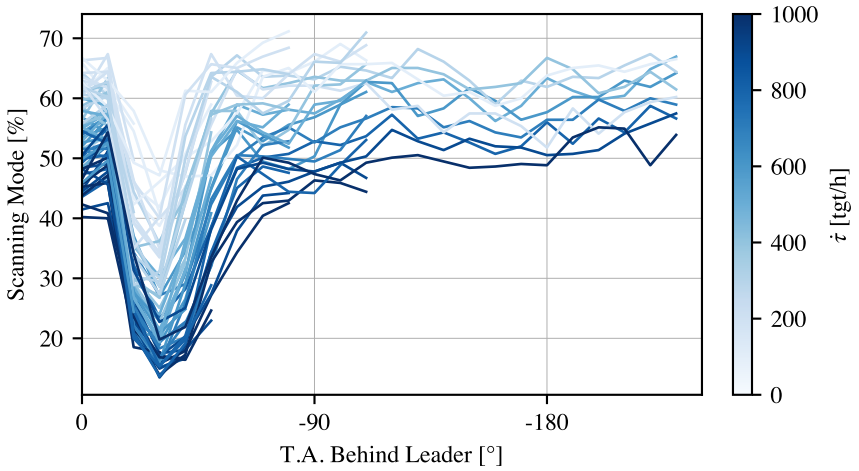


Figure 7.10: Scanning time fraction of each satellite in various String constellations.

7.4.3.1 Scanning vs. Imaging Frequency

In the String constellation, a satellite's position in the string is predicted to influence the behaviors exhibited by that satellite. Figure 7.10 tests this hypothesis by plotting the amount of time spent in the scanning mode versus the satellite's position in the string. The leader scans 40% to 68% of the time, since known targets are sparse in the upcoming ground track. The fourth to sixth satellite spends the smallest fraction of time (13% to 47%) in the scanning mode, since the leading satellites have found a dense list of targets to be scanned. As one looks farther back in the string, scanning fractions increase since most possible targets have been scanned and imaged, so these followers can only find newly appearing targets.

Differing scanning behavior is also expected among agents in the Ring constellation, but due to the constellation's symmetry, a bias toward scanning versus imaging is expected to change over time. Figure 7.11 shows the orbit-by-orbit scanning percentage of each agent in an $S = 6$ Ring constellation. While initially the behaviors are homogenous for the first two orbits, diversification of behaviors is first evident for orbits two through five, where the even-indexed satellites take a scanning-heavy role, while odd-indexed satellites tend toward imaging. This behavior inverts—though less prominently—during orbits six through eight. In essence, the constellation has dynamically created tip-and-cue pairs, as needed. Through the end of the episode, scanning is less

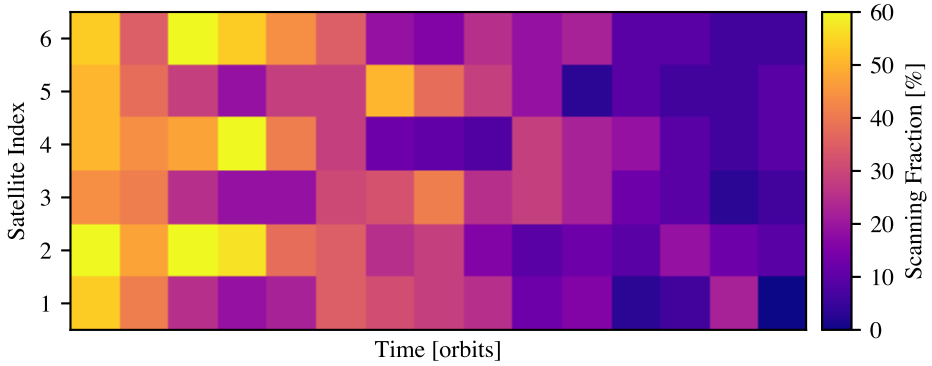


Figure 7.11: Scanning time fraction of each satellite in a Ring constellation. Target rate $\dot{\tau} = 1000$ tgt/h.

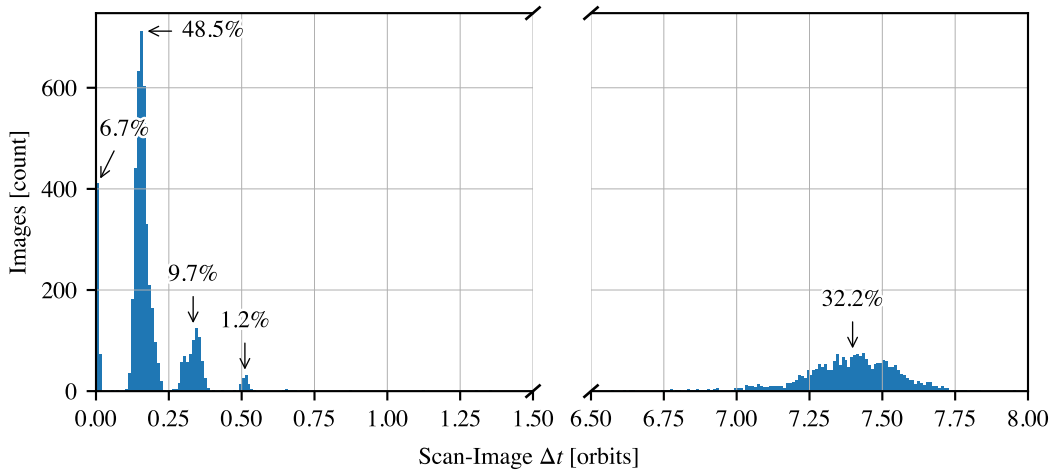


Figure 7.12: Histogram of delay between scanning and imaging in the Ring constellation with $S = 6$ satellites.

frequent throughout the constellation since many targets are available to image that were scanned but not imaged during earlier opportunities.

7.4.3.2 Scan-Image Delays

The final evidence for collaboration is that the satellite that scans a given target is typically not the satellite that images it. Figure 7.12 shows a histogram of the scan-image delay for all imaged targets in a Ring constellation with $S = 6$ and $\dot{\tau} = 1000$ targets per hour. On the left side of the plot, each grouping is a different follower satellite imaging the scanned target. As such, only

6.7% of targets are imaged by the same satellite that scanned it, while 59.4% are imaged by one of the next three following satellites (at which point Earth's rotation has moved the target out of the ground track). The remainder of images are taken a half-day later when the satellites have another opportunity to image them; at this point, the specific agent that images a target is irrelevant. Satellites are working together because the majority of images are collected with minimal delay by a later satellite.

7.5 Realistic Test Case

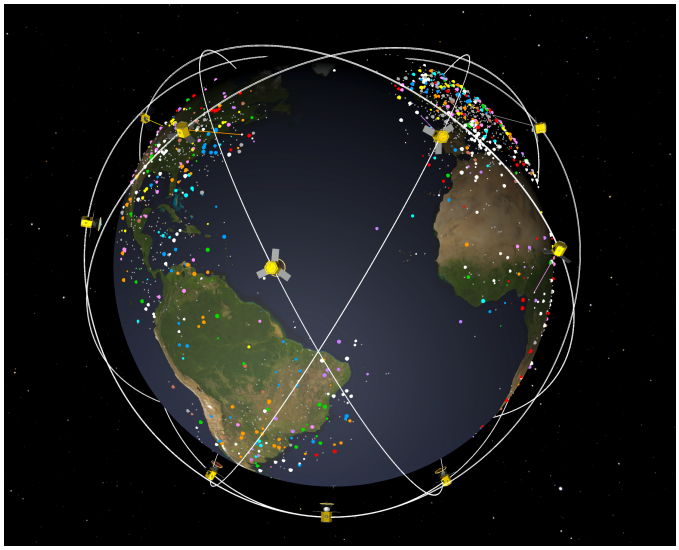


Figure 7.13: The Walker-Delta constellation with clustered targets.

The performance of the policy is further evaluated in realistic test cases. Three factors are considered: 1) using of a Walker-Delta constellation [135]; 2) limiting communication to line-of-sight (LOS) connections between satellites; and 3) having a non-uniform target distribution. The Walker-Delta constellation has 18 satellites, with 6 evenly-phased planes of 3 satellites each at a 60° inclination. This constellation has limited in-plane interaction but significant between plane interaction, testing a quality not seen in other benchmarks. For the non-uniform clustered target distribution, target locations are randomly selected in a region around population centers[3] , yielding

[3] City location data from simplemaps.com, Basic World Cities Database v1.901, CC BY 4.0.

a clustered distribution of points. Each combination of free or LOS communication and uniform or clustered targets are evaluated for $\dot{\tau} = 250$ tgt/h across five seeds. The Walker-Delta constellation is depicted inspecting the clustered target distribution in Figure 7.13.

Table 7.3: Performance of Ring-6 policy in the realistic scenario ($\mu \pm \sigma$).

Targets	**Uniform**		**Clustered**	
Comm.	**Free**	**LOS**	**Free**	**LOS**
Reward ΣR	2496 ± 8	2507 ± 12	2250 ± 31	2246 ± 30
$\Sigma R / S\dot{\tau}$	0.55 ± 0.00	0.55 ± 0.00	0.50 ± 0.00	0.49 ± 0.01
Scan Mode %	63.6 ± 0.5	63.4 ± 0.5	76.0 ± 0.3	75.8 ± 0.7
Known $\mathcal{K}_{\%}$	98.1 ± 0.3	98.2 ± 0.3	85.5 ± 0.8	85.0 ± 0.5
Imaged $\mathcal{I}_{\%}$	95.6 ± 0.3	95.6 ± 0.2	77.3 ± 0.6	77.0 ± 0.7
Rel. Ring-1	1.25 ± 0.01	1.24 ± 0.02	1.12 ± 0.03	1.12 ± 0.01

Table 7.3 summarizes the results of these test cases, using metrics previous analyzed in Figure 7.4 and Figure 7.9. Comparing the Free and LOS columns, a major result is that deploying the policy in an environment with more realistically limited communication does not significantly change the behavior or performance of the policy. The last row, giving the total reward of the String-6 policy versus the String-1 policy in the same test cases, again shows that altruistic and collaborative behaviors have been learned. This indicates that the policy generalizes well to a scenario where out-of-plane interactions are common, even though only in-plane interaction was experienced in training.

Chapter 8

The Resident Space Object Inspection Problem

8.1 Motivation

With the proliferation of satellites in low Earth orbit (LEO), rendezvous and proximity operations (RPO) are becoming increasingly important. These include servicing and interacting with active spacecraft, deorbiting defunct satellites, identifying damage on assets, and characterizing unfamiliar objects. For these operations, the resident space object (RSO) must be inspected to determine docking points, damage, or other properties. This inspection task is complex: the servicer is required to maneuver around the RSO to inspect all illuminated surfaces while avoiding collision. Currently, close-proximity operations are operationally challenging and require significant ground support to upload open-loop command sequences and monitor the resulting performance. Close-proximity relative motion maneuvers must be fuel efficient, avoid collisions, and satisfy imaging requirements.

A variety of path planning methods have been proposed for the inspection task. References 43, 76, 77 demonstrate pipelines for global planning of inspection trajectories for a swarm of satellites, for impulsive and continuous thrust control. These plan a set of stable relative orbits that should provide coverage of the RSO, then assign orbits to individual servicers within the swarm, calculating optimal transitions between orbits. Reference 44 decomposes a large RSO into primitive shapes and projects inspection paths onto them. Such two-phase methods introduce significant suboptimality by constraining the solution space to certain classes of trajectories. Even among methods that are less constrained and thus more optimal, solutions tend to fight the natural relative motion

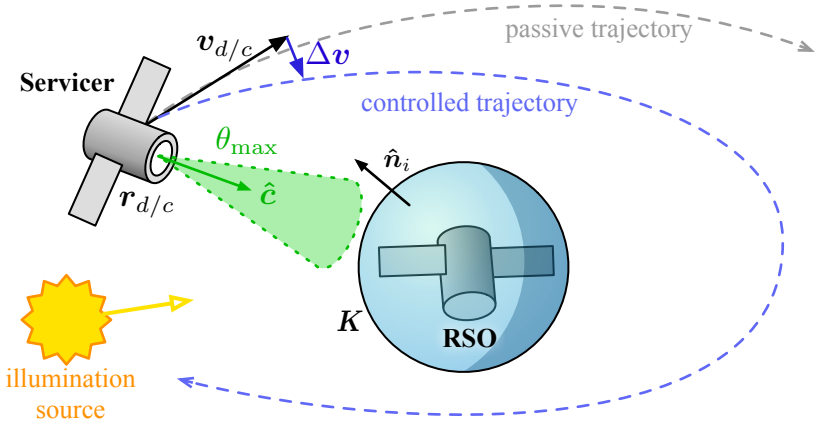


Figure 8.1: The RSO inspection task with a single servicer.

dynamics: Across the literature, fast ($<$ 1 orbit) and high-Δv cost (dozens of m/s) inspection trajectories are generated that largely counteract any natural motion [54].

The research in the following chapters aims to find a fuel-efficient class of solutions for the RSO inspection problem with impulsive-thrusting servicers that leverage natural relative motion present in LEO. To this end, this chapter develops the RSO inspection problem depicted in Figure 8.1 for RSO in an eccentric orbit with a Hill-frame-fixed attitude and one or more servicers, subject to range, pointing, and lighting constraints for inspection.

8.2 Problem Statement

In the inspection task, a servicer spacecraft has the objective of inspecting all facets of an RSO in LEO, subject to illumination constraints. To achieve this, the servicer must use discrete thrusts to control its motion relative to the RSO while avoiding collisions between the two spacecraft.

8.2.1 Satellite Dynamics

The RSO (typically the chief) is a non-maneuvering satellite in an eccentric LEO orbit with a Hill-frame-fixed, nadir-pointing attitude; this attitude assumption is reflective of the attitude control for many active LEO satellites. The origin of this Hill/body frame $\mathcal{H}$ is used for all

displacement vectors unless otherwise noted. The RSO's orbit is a randomly sampled LEO orbit in each instance of the environment. The altitude of apogee and perigee are uniformly sampled between 500 and 1100 km, and the angular orbital elements are uniformly sampled in their respective domains. Results for the subsequent chapters generalize at least within these bounds.

The servicer (typically the deputy) is an actively maneuvering satellite operating near the RSO. The Hill-frame state of deputy d relative to the chief c is written as the position $\boldsymbol{r}_{d/c}$ and velocity $\boldsymbol{v}_{d/c}$. The servicer is equipped with a body-fixed camera $\hat{\boldsymbol{c}}$ to inspect the RSO, and the attitude is actively controlled to point the instrument at the RSO (i.e., $\hat{\boldsymbol{c}}$ is kept antiparallel to $\boldsymbol{r}_{d/c}$). The servicer can control its motion with a thruster that can produce impulsive thrusts $\Delta\boldsymbol{v}$ of magnitude up to $\Delta v_{\max} = 1$ m/s in an arbitrary direction, followed by a passive drift period. Because the period between thrusts tends to be long, this could be achieved by a variable-thrust thruster on an actuated platform that reorients between maneuvers. At most, each servicer may use 10 m/s of Δv to complete the task.

To avoid collision, the deputy must stay out of the chief's keep-out ellipsoid represented by a 3×3 positive definite matrix $\boldsymbol{K}_{d/c}$ consisting of the reciprocal squared semi-axes with a constraint in the general form

$$\boldsymbol{r}_{d/c}^T(t)\boldsymbol{K}_{d/c}\boldsymbol{r}_{d/c}(t) \geq 1 \tag{8.1}$$

For a spherical keep-out region of radius $R_{d/c}$ (i.e., $\boldsymbol{K}_{d/c} = \boldsymbol{I}/R_{d/c}^2$), as is primarily considered in this work, this constraint reduces to

$$\|\boldsymbol{r}_{d/c}(t)\| \geq R_{d/c} \tag{8.2}$$

Analysis of the relative motion dynamics of the system is necessary for the shield and for the probabilistic roadmap (PRM)-based solution. The motion of the servicer can be described with a relative motion state transition matrix for the unperturbed motion of a deputy spacecraft relative to a chief spacecraft in an eccentric orbit. In the Hill frame $\mathcal{H}$ of the chief satellite C (which is the same as its body frame due to the Hill-fixed attitude assumption), the state transition matrix for the state vector $\boldsymbol{x}_{d/c} = \begin{bmatrix} {}^{\mathcal{H}}\boldsymbol{r}_{d/c}^T & {}^{\mathcal{H}}\boldsymbol{v}_{d/c}^T \end{bmatrix}^T$ composed of the Hill-frame position and velocity of the

deputy relative to the chief is given by

$$\boldsymbol{\Phi}_c^x(t,t_0) = \boldsymbol{A}_c(t)\boldsymbol{\Phi}_c^{\delta\boldsymbol{\alpha e}}(t,t_0)\boldsymbol{A}_c^{-1}(t_0) \tag{8.3}$$

where $\boldsymbol{A}_c(t)$ is the linearized relative orbital element to Hill frame mapping matrix, and $\boldsymbol{\Phi}_c^{\delta\boldsymbol{\alpha e}}(t,t_0)$ is the relative orbital element state transition matrix.[1] Left superscripts are used to denote the frame the vector is expressed in.

8.2.2 Inspection Task

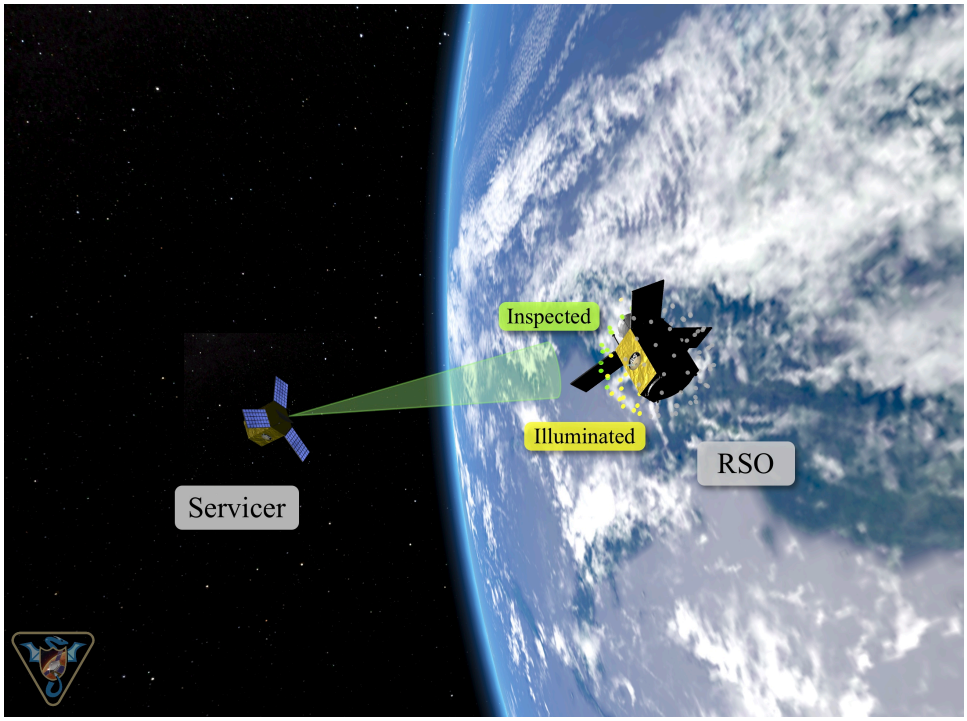


Figure 8.2: Rendering of the RSO inspection task with inspection points.

For the inspection task, a set of required inspection points $\mathcal{P}$, the criteria for inspecting them, and the criteria for task completion are established.

The RSO has a set of body-fixed inspection point structures $p \in \mathcal{P}$, each with an associated body-fixed position $\boldsymbol{r}_p$ (relative to the RSO origin) and normal vector $\hat{\boldsymbol{n}}_p$, such that $p_p = (\boldsymbol{r}_p, \hat{\boldsymbol{n}}_p)$. In this work, the RSO's geometry is modeled as a 1-meter radius sphere of $|\mathcal{P}| = 100$ uniformly

[1] See Appendix A for $\boldsymbol{A}_c$, $\boldsymbol{A}_c^{-1}$, and $\boldsymbol{\Phi}_c^{\delta\boldsymbol{\alpha e}}$ from Reference 75.

distributed points with surface normals in the radial direction, as seen in Figure 8.2. Points are illuminated when the angle between the Sun vector $\boldsymbol{r}_{\text{sun}}$ and the point normal is less than $\phi_{\max} = 60°$ and the RSO is not being eclipsed by the Earth. Illumination can be reduced to a function of the RSO's mean anomaly M:

$$L(p_p, M) = \neg\text{Eclipse}(p_p, M) \wedge \angle\left(\hat{\boldsymbol{n}}_p, \boldsymbol{r}_{\text{sun}}(M) - \boldsymbol{r}_p\right) < \phi_{\max} \tag{8.4}$$

The set $\mathcal{L}$ of points that are illuminated for at least some time during the orbit is thus defined as

$$\mathcal{L} = \{p_p \in \mathcal{P} \mid \exists M \in [0, 2\pi) \text{ such that } L(p_p, M) = \text{True}\} \tag{8.5}$$

Figure 8.3 shows the percent of the RSO that is illuminated over 1000 random cases as a function of the β angle, the angle between the orbital plane and the Sun vector. Orbits that are closer to a Sun-synchronous orbit (SSO) at the dusk-dawn boundary tend to have the smallest fraction of points ever illuminated, as the orbit-frame-fixed attitude leads to one side of the spacecraft constantly facing away from the Sun.

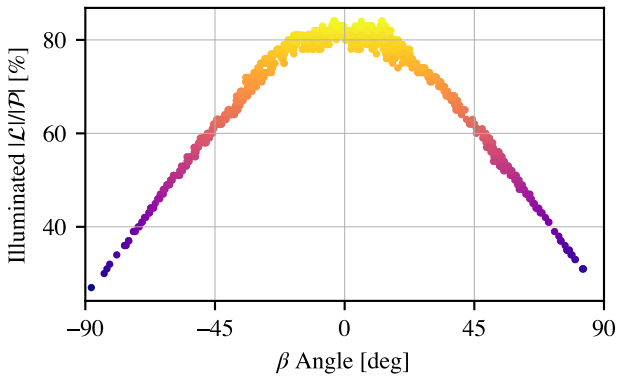


Figure 8.3: Fraction of RSO illuminated as a function of orbital β angle.

To image an inspection point, the servicer must be in range of the point, the instrument must have an acceptable relative angle, and the point must be illuminated. The range constraint requires that the servicer is within $R_{\max} = 250$ m of the point p_p being inspected:

$$\|\boldsymbol{r}_{d/c}(M) - \boldsymbol{r}_p\| < R_{\max} \tag{8.6}$$

The squint angle constraint requires that the servicer's instrument views the point p_p from an angle no more than $\theta_{\max} = 30°$

$$\angle(\hat{\boldsymbol{n}}_p, \boldsymbol{r}_{d/c}(M) - \boldsymbol{r}_p) < \theta_{\max} \tag{8.7}$$

and the illumination constraint requires that $L(p_p, M)$ is True. Combined, these constraints define an inspectability indicator function

$$I(p_p, \boldsymbol{r}_{d/c}, M) = \text{Equation 8.6} \wedge \text{Equation 8.7} \wedge L(p_p, M) \tag{8.8}$$

that is a function only of mean anomaly and servicer position.

When Equation 8.8 is satisfied for some point p, the point is added to the inspected set $\mathcal{I}$. The RSO is considered fully inspected and the task is complete when at least 90% of the points in $\mathcal{L}$ are in $\mathcal{I}$. This threshold is set because some points on the limb of the spacecraft may only be instantaneously illuminated. The complete problem is then a constrained multiobjective optimization problem: minimize the fuel consumed and time taken while completing the inspection task.

8.2.3 Multi-Agent Inspection

The problem is sometimes considered for multiple servicers, indexed as $d_1, ..., d_N$. The goal of the problem remains the same: jointly inspect at least 90% of the illuminated points $\mathcal{L}$. It is reasonably assumed that servicers can communicate freely with each other due to their ≤ 2 km proximity.

The biggest challenge introduced by having multiple servicers is avoiding collisions between any pair of satellites in the environment. A keepout constraint is introduced between each pair of servicers d_i and d_j, represented by the ellipsoid $\boldsymbol{K}_{d_i/d_j}$. As with Equation 8.1, the constraint is in the form

$$\boldsymbol{r}_{d_i/d_j}^T(t)\boldsymbol{K}_{d_i/d_j}\boldsymbol{r}_{d_i/d_j}(t) \geq 1 \tag{8.9}$$

and reduces to

$$\|\boldsymbol{r}_{d_i/d_j}(t)\| \geq R_{d_i/d_j} \tag{8.10}$$

for a spherical keepout of radius R_{d_i/d_j}. Because defining a keepout ellipsoid in the RSO Hill frame for servicer-servicer collisions rarely makes sense due to the servicers' time-varying attitudes, the spherical keepout is almost always used in practice.

8.3 Simulation Environment

As with the agile Earth-observing satellite scheduling problem (AEOSSP), the RSO inspection problem is modeled with a high-fidelity simulation implemented in BSK-RL (chapter 2). As a result, it provides an apples-to-apples comparison environment for the two solution methods presented in the next chapters. It also tests the performance of these methods outside idealized dynamical conditions: approximations and linearizations utilized by the methods (such as Clohessy-Wiltshire-Hill (CWH) equations or the linearized relative orbit element transformation in Appendix A) face perturbations such as J2 and atmospheric drag. A visualization of the environment is shown in Figure 8.2.[2]

[2] Visualizations of select cases are available at https://www.youtube.com/watch?v=eQEoTOYADKc.

Chapter 9

Autonomous Inspection with Reinforcement Learning

9.1 Motivation

Reinforcement learning (RL) has been identified as a promising method for closed-loop autonomy for the resident space object (RSO) inspection task. In Reference 80, a neural network (NN)-based policy is used for high-level planning between predetermined inspection waypoints. Van Wijk [163] approaches the problem with a continuous action space for continuous, low-thrust control, considering lighting conditions. In Lei's work [81, 82], the waypoint-based approach is used for distributed and centralized control of a swarm of servicers. Brandonisio [83, 84] and Quinn [164] consider the problem with a simultaneous localization and mapping (SLAM) component. However, a drawback of this class of methods is a lack of safety guarantees. While penalties can be included to disincentivize unsafe actions, there is no guarantee that the agent will not take an unsafe action.

To address this challenge, a variety of methods for bounding the performance and guaranteeing safety of RL-based policies have been developed [165]. Several references combine RL with trajectory optimization steps: References 85 and 86 leverage the transformer architecture for rendezvous and proximity operations (RPO), while Majumdar [166] uses RL with traditional path planning methods as a way of directing the search of the traditional methods while maintaining robustness. Shielded RL [55] has been demonstrated as an effective method for various other spacecraft tasking problems, including resource management [58], small-body proximity operations [79], and agile Earth observation under various conditions [104, 100]. Control barrier functions (CBFs) are an alternative to shields that provide guarantees on performance: Most similar to this chapter,

papers by Van Wijk and Dunlap [54, 87] ensure safety in inspection tasks controlled by RL with continuous control inputs using a CBF. Their safety constraints include the keep-out zone present in this work, fixed-horizon passive safety, velocity constraints, a Sun pointing constraint, and a keep-in zone. However, across the literature high fuel costs on the order of 10's of m/s are observed as solutions largely counteract natural relative motion rather than leverage it.

To develop safety guarantees for these tasks, existing work on safe coordinated motion planning in low Earth orbit (LEO) can be utilized. Morgan [71, 72] uses sequential convex programming (SCP) to plan collision-free continuous thrust trajectories. References 73 and 74 use analytical methods to derive swarm reconfiguration laws with impulsive thrusts in terms of relative orbital elements.

This chapter uses RL to find closed-loop fuel-efficient trajectories for the RSO inspection problem. A single- and multi-agent Markov decision process (MDP) is formulated for the impulsive-thrust-and-drift control scheme and policies are trained for a range of values on the fuel-time Pareto front. Shields are derived and validated that guarantee action safety in a two-satellite circular-chief case and a more general N-satellite eccentric-chief case. Finally, two methods for multi-agent inspection are investigated.

9.2 MDP Formulation

The single-servicer inspection task is formalized as a partially-observable semi-Markov decision process (POsMDP) [26, 147, 162]. The elements of the MDP tuple for the inspection task are as follows:

- **State Space:** The state of the simulator providing the generative model for the MDP. This includes satellite dynamic states, flight software (FSW) states, and environment states. Terminal states for the servicer are encountered when any of the following conditions are met:
  - $\geq 90\%$ of illuminated points are inspected, $|\mathcal{I}|/|\mathcal{L}| \geq 0.9$.

Table 9.1: Elements of the observation space for each servicer.

Element	Dim.	Description
${}^{\mathcal{H}}\boldsymbol{r}_{d/c}$	3	RSO-relative position
${}^{\mathcal{H}}\boldsymbol{v}_{d/c}$	3	RSO-relative velocity
${}^{\mathcal{H}}\hat{\boldsymbol{s}}$	3	Sun direction
${}^{\mathcal{S}}\boldsymbol{e}$	3	orbital eccentricity vector
${}^{\mathcal{S}}\boldsymbol{h}$	3	orbital angular momentum vector
${}^{\mathcal{S}}\boldsymbol{r}_{DE}$	3	orbital position
Δv_{avail}	1	remaining available Δv
t	1	time since start of episode
$[\tau^o_{\text{ecl}}, \tau^c_{\text{ecl}}]$	2	next eclipse start and end times
$\%_{\text{inspect}}(\mathcal{P})$	M	region inspection status

  - The servicer collides with the RSO, in violation of Equation 8.1.
  - The servicer leaves the region surrounding RSO: $\|\boldsymbol{r}_{d/c}\| > 1$ km. This condition is included to encourage "good" behavior when training, but is not a hard constraint enforced by the shield when using the policy.
  - The servicer runs out of available fuel: $\Delta v_{\text{avail}} = 0$.
  - The episode times out: $t \geq 10$ RSO orbits.

- **Observation Space:** The observation for the servicer is composed of a subset of the elements of the state space and transformations thereof. The elements of the observation space are given in Table 9.1, which are then normalized to be on a similar order of magnitude. Two reference frames are used: the RSO Hill/body frame $\mathcal{H}$, and the Earth-Sun Hill frame $\mathcal{S}$.

  The region inspection status $\%_{\text{inspect}}(\mathcal{P})$ is a compact way of representing what points on the RSO have been inspected, rather than a vector of $|\mathcal{P}|$ Booleans. For this observation, inspection points are grouped into M clusters. Each element of the observation is the frac-

tion of points in the corresponding cluster that have been inspected. In this environment, $M = 15$ clusters are evenly spaced on the surface of the sphere, resulting in regions roughly the same size as the servicer's field of view (FOV). This observation does not explicitly tell the agent which points will be illuminated during the orbit; the agent must infer that from the observations of the orbital state.

- **Action Space:** The servicer's action space is $a \in \Delta v_{\max} \mathcal{B}^3 \times [0, \Delta t_{\max}]$. The first three elements specify the direction and magnitude of an impulsive thrust within a ball, and the last element specifies the duration to drift before executing the next thrust. This continuous action space is in contrast to prior work that uses a flight mode-based approach [162] and ongoing work that considers hybrid action spaces [134].

- **Reward Function:** The reward $R(s, s')$ for the servicer is the sum of five functions. First, reward is yielded for inspecting points on the RSO that have not yet been inspected by any servicer.

$$R_{\text{inspection}}(s, s') = \frac{|\mathcal{I}'| - |\mathcal{I}|}{|\mathcal{P}|} \tag{9.1}$$

A reward is also given to all agents for jointly reaching the goal state of 90% inspection of illuminated points.

$$R_{\text{success}}(s') = \beta_{\text{success}} \quad \text{if } \frac{|\mathcal{I}'|}{|\mathcal{L}|} \geq 90\% \tag{9.2}$$

A penalty for fuel use is given at the time of a burn, weighted by the fuel use penalty α.

$$R_{\text{fuel}}(\Delta \boldsymbol{v}) = -\alpha \|\Delta \boldsymbol{v}\| \tag{9.3}$$

A penalty for time use is given per-step, weighted by the fuel use penalty α_t.

$$R_{\text{time}}(\Delta t) = -\alpha_t \Delta t \tag{9.4}$$

Finally, a failure penalty is given to a servicer that reaches any of the negative terminal states (collision, running out of fuel, or leaving the region of space around the RSO)

$$R_{\text{failure}}(s') = -\beta_{\text{fail}} \quad \text{if } s' \in \mathcal{S}_{\text{fail}} \tag{9.5}$$

Table 9.2: Additional observations for servicer d_i included in some multi-agent scenarios.

Element	Dim.	Description
${}^{\mathcal{H}}\boldsymbol{r}_{d_j/c}\ \forall d_j \neq d_i$	$3(N-1)$	RSO-relative positions of other servicers
${}^{\mathcal{H}}\boldsymbol{v}_{d_j/c}\ \forall d_j \neq d_i$	$3(N-1)$	RSO-relative velocities of other servicers

Rewards are yielded at the end of a semi-Markov decision process (sMDP) step. This is an approximation in the case of the incremental inspection reward, as it would be more appropriate to have a density function that distributes rewards over the course of the step. With a discount rate close to 1, the difference is minimal.

- **Transition Model:** Instead of a probabilistic transition function, the transition model is implemented as a deterministic generative model. When an action is taken, the environment propagates the simulation forward in time until the next action is to be taken. The sMDP Δt is the duration for which the simulator propagated the step, as selected by the action. Collisions and other failure states can also trigger the end of a step prematurely.

When considering multiple servicers, the formulation is extended to an asynchronous decentralized partially-observable semi-Markov decision process (Dec-POsMDP). The following modifications are made to the MDP:

- The inspection objective is jointly completed by all servicers. The region inspection status $\%_{\text{inspect}}$ observation reflects the inspection completed by any servicer. Incremental inspection rewards are given on a per-servicer basis, but the completion bonus is yielded for all servicers once the success threshold is met.

- An additional terminal state is added for servicer-servicer collisions, reflecting the constraint of Equation 8.9.

- Actions are taken asynchronously by each servicer, and a new action for a given servicer is only selected when the previous action has completed.

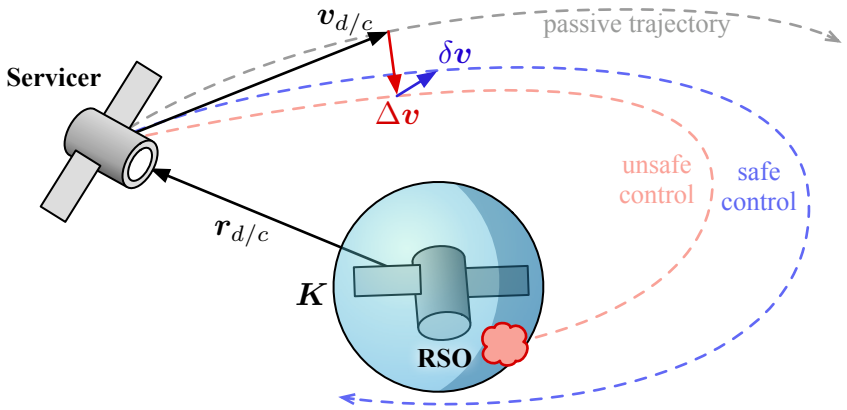


Figure 9.1: The shield guarantees safety when the servicer maneuvers.

- When specified (i.e. for policies $\pi_{\text{multi},\alpha}$), the additional observations about other agents' states described in Table 9.2 are included in each agent's observation.

9.3 Guaranteeing Relative Motion Safety With Shields

Collision with other spacecraft is of high concern for the servicer spacecraft. Shielding is a method for guaranteeing operational safety of a RL-based agent by preventing the execution of actions that are known to be unsafe. For this problem, a shield is developed that prevents the servicer from executing a burn that may lead to eventual collision with another spacecraft under natural dynamics.

A shield interacts with a policy in order to prevent unsafe actions (i.e., those actions that may cause a transition into $s' \in S_{\text{unsafe}}$ in the next step, or for more robustness, in any future step) [55]. The shield in this work uses the action projection approach to shielding [165]: if the desired action falls in the space of disallowed actions, the closest safe action is selected instead.

The conceptual behavior of the shield is shown in Figure 9.1. When a policy selects some $\Delta\boldsymbol{v}$, the shield determines the smallest deviation $\delta\boldsymbol{v}$ that makes the new trajectory passively safe over an infinite horizon under relative motion dynamics. Since the servicer is already known to be on a passively safe trajectory from the previous iterations of shielded tasking, there is always at least the trivial solution $\delta\boldsymbol{v} = -\Delta\boldsymbol{v}$ that satisfies the safety constraints by keeping the servicer in

the same passively safe orbit as before. This infinite-horizon safety also means that a missed firing for any reason does not jeopardize safety. As a result of the shield, the servicer is never on a direct collision course that depends on an additional future burn to maintain safety.

In this chapter, two shield designs are studied. The first considers a simpler case, with one servicer, spherical keepout regions, and the RSO in a circular orbit. The second extends the shield to account for multiple servicers, elliptical keepout regions, and an eccentric orbit. For the second case, a more efficient method of computing the shield is also developed.

9.3.1 Single Servicer, Circular Orbits

First, the shield formulation is derived for a single servicer and an RSO in a circular orbit. The keepout region is also assumed to be spherical.

9.3.1.1 CWH Dynamics

For this shield, the relative motion of the servicer is analyzed under Clohessy-Wiltshire-Hill (CWH) dynamics. The CWH equations of relative motion are a set of linearized equations that describe the unperturbed relative motion of a deputy spacecraft relative to a chief spacecraft in a circular orbit. In the Hill frame, the CWH equations are given by

$$\dot{\boldsymbol{x}}_{d/c} = \boldsymbol{A}\boldsymbol{x}_{d/c},\ \boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 0 & 2n & 0 \\ 0 & 0 & 0 & -2n & 0 & 0 \\ 0 & 0 & -n^2 & 0 & 0 & 0 \end{bmatrix} \tag{9.6}$$

where the state vector

$$\boldsymbol{x}_{d/c} = \begin{bmatrix} \mathcal{H}\boldsymbol{r}_{d/c}^T \\ \\ \mathcal{H}\boldsymbol{v}_{d/c}^T \end{bmatrix} \tag{9.7}$$

is composed of the Hill-frame position and velocity of the deputy relative to the chief, and $n = 2\pi/T_{\text{orbit}}$ is the mean motion of the chief.

9.3.1.2 Optimization Problem

With these dynamics, the shield can be formulated as an optimization problem. In short, the problem is to find the thrust closest to that selected by the policy $\pi(s) = \Delta\boldsymbol{v}$ that does not lead to collision with the RSO under CWH dynamics. Let $\delta\boldsymbol{v}$ be the change in selected thrust required for safety. The optimization problem can be written as

$$\text{minimize} \quad \|\delta\boldsymbol{v}\| \tag{9.8}$$

$$\text{such that} \quad \|\Delta\boldsymbol{v} + \delta\boldsymbol{v}\| \leq \Delta v_{\max} \tag{9.9}$$

$$\boldsymbol{x}_0 = \begin{bmatrix} \boldsymbol{r} \\ \\ \boldsymbol{v} + \Delta\boldsymbol{v} + \delta\boldsymbol{v} \end{bmatrix}$$

$$\boldsymbol{x}_0^T \exp(\boldsymbol{A}t_i)^T \begin{bmatrix} I & 0 \\ \\ 0 & 0 \end{bmatrix} \exp(\boldsymbol{A}t_i)\boldsymbol{x}_0 \geq (R_{d/c} + \epsilon)^2 \quad \forall t_i \in [t_0, t_{\max}] \tag{9.10}$$

$$\pm\left((R_{d/c} + \epsilon) + 2A_0' \pm y_{\text{off}}'\right)x_{\text{off}}' \leq \frac{3}{2} t_{\max} n x_{\text{off}}'^2 \tag{9.11}$$

The objective (Equation 9.8) minimizes the change to the control requested by the policy. The first constraint (Equation 9.9) ensures that the thrust does not exceed the maximum possible thrust. The second constraint (Equation 9.10) ensures that the shielded action does not lead to collision with the RSO over some upcoming timespan $[t_0, t_{\max}]$. An additional safety parameter ϵ is added to the collision constraint to account for the effects of perturbations or an additional safety factor.

The final two constraint equations (Equation 9.11) are necessary for infinite-horizon passive safety on the interval $[t_{\max}, \infty)$, and thus could be neglected for weaker safety guarantees. With this constraint, the servicer must either be in a non-colliding periodic orbit or on a secularly moving-away trajectory at $t_{\max}$ after the initial time; these conditions are derived in the next section. This constraint uses the standard definitions of x'_{off}, y'_{off}, and A'_0 [75], where $'$ indicates values calculated using the state after $\delta\boldsymbol{v}$ has been applied.

Since noncollision cannot be checked continuously for all times in $[t_0, t_{\max}]$, the constraint is enforced at times on the interval discretized by Δt. The discretization interval must be sufficiently small to prevent collisions between timesteps. The condition for this is similar to the one found by Morgan [71]. With an upper bound on relative velocity of the servicer estimated as

$$\lceil v \rceil = \sqrt{(A_0 n)^2 + (2A_0 n - \frac{3}{2} n x_{\text{off}})^2 + (B_0 n)^2} + \Delta v_{\max} \tag{9.12}$$

the maximum undetectable violation of the collision constraint for some Δt is given by

$$r_{\text{violation}} \leq \frac{\Delta t \lceil v \rceil}{2} \tag{9.13}$$

It follows that a maximum allowable violation $r_{\text{violation}} < \epsilon$ can be specified and used to select Δt.

9.3.1.3 Conditions for Infinite-Horizon Safety

To derive the infinite-horizon passive safety constraint enforced by Equation 9.11, the conditions for which no collisions are possible under forward propagation of the relative motion dynamics from $t_{\max}$ must be identified. This passive safety case occurs when there is secularly increasing separation or periodic motion between the servicer and the RSO.

The general solution to the CWH equations (Equation 9.6) in the Hill frame are given by

$$x(t) = A_0 \cos(nt + \alpha) + x_{\text{off}} \tag{9.14}$$

$$y(t) = -2A_0 \sin(nt + \alpha) - \frac{3}{2} n t x_{\text{off}} + y_{\text{off}} \tag{9.15}$$

$$z(t) = B_0 \cos(nt + \beta) \tag{9.16}$$

Noting that only the y component (Equation 9.15) has a secular term, two cases for passive safety are identified:

(1) The relative motion is periodic and safe. In this case, the secular term coefficient must be zero to ensure periodicity (up to any perturbations):

$$-\frac{3}{2}nx_{\text{off}} = 0 \implies x_{\text{off}} = 0 \tag{9.17}$$

The safety of the periodic motion is guaranteed as long as it is safe over the course of one orbit. In the optimization problem, the collision constraint (Equation 9.10) checks for this requirement as long as the period is sufficiently long,

$$t_{\max} - t_0 \geq T_{\text{orbit}} \tag{9.18}$$

(2) Alternatively, the secular term is nonzero and dominates the motion. In this case, the magnitude of the local extrema around $t_{\max}$ must be greater than the keepout radius and be growing with t:

$$\left| -\frac{3}{2}nt_{\max}x_{\text{off}} \pm 2A_0 + y_{\text{off}} \right| \geq R_C + R_D + \epsilon \tag{9.19}$$

$$\frac{d}{dt}\left| -\frac{3}{2}ntx_{\text{off}} \pm 2A_0 + y_{\text{off}} \right|_{t=t_{\max}} > 0 \tag{9.20}$$

It is worth noting that this constraint on $y(t)$ is more strict than necessary, as it is possible that the periodic motion of $x(t)$ and $z(t)$ are enough by themselves to prevent collision with arbitrary behavior in $y(t)$.

The constraints, which can be expressed logically as

$$\text{Equation 9.17} \vee (\text{Equation 9.19} \wedge \text{Equation 9.20}) \tag{9.21}$$

reduce to

$$\frac{3}{2}t_{\max}nx_{\text{off}}^2 \geq -x_{\text{off}}\left((R_C + R_D + \epsilon) + 2A_0 + y_{\text{off}}\right) \tag{9.22}$$

and

$$\frac{3}{2}t_{\max}nx_{\text{off}}^2 \geq -x_{\text{off}}\left(-(R_C + R_D + \epsilon) - 2A_0 + y_{\text{off}}\right) \tag{9.23}$$

which can be combined into Equation 9.11.

9.3.2 Multiple Servicers, Elliptical Orbits

Using a similar structure, the shield is extended to account for multiple servicers, ellipsoidal keepout regions, and eccentric RSO orbits. The shield enforces passive safety relative to all other satellites in the environment.

9.3.2.1 Optimization Problem

Again, the shield can be formulated as an optimization problem. The problem is to find the thrust for spacecraft d_i closest to that selected by its policy $\pi(s) = \Delta\boldsymbol{v}$ that does not lead to collision with the RSO or any other satellite under the first-order eccentric relative motion dynamics, as described in Appendix A. Unlike the CWH equations, these dynamics account for the effects of an eccentric chief. In particular, the state transition matrix (STM) for motion relative to a chief in an eccentric orbit $\boldsymbol{\Phi}_c^x$ is utilized.

Let $\delta\boldsymbol{v}$ be the change in selected thrust required for safety. The optimization problem for servicer d_i can be written as

$$\text{minimize} \quad \|\delta\boldsymbol{v}\| \tag{9.24}$$

$$\text{such that} \quad \|\Delta\boldsymbol{v} + \delta\boldsymbol{v}\| \leq \Delta v_{\max} \tag{9.25}$$

$$\text{for each } c^* \in \{c, d_1, ..., d_N | c^* \neq d_i\}$$

$$\boldsymbol{x}_{d_i/c^*}(t_0) = \begin{bmatrix} {}^{\mathcal{H}}\boldsymbol{r}_{d_i/c^*} \\ {}^{\mathcal{H}}(\boldsymbol{v}_{d_i/c^*} + \Delta\boldsymbol{v} + \delta\boldsymbol{v}) \end{bmatrix}$$

$$\boldsymbol{x}_{d_i/c^*}(t) = \boldsymbol{\Phi}_{c^*}^x(t, t_0)\boldsymbol{x}_{d_i/c^*}(t_0)$$

$$\left(\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \boldsymbol{x}_{d_i/c^*}(t)\right)^T \boldsymbol{K}_{d_i/c^*} \left(\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \boldsymbol{x}_{d_i/c^*}(t)\right) > 1 \quad \forall\ t \in [t_0, t_{\max}] \tag{9.26}$$

$$g_{\text{low,high}}\left(\boldsymbol{oe}(t_{\max}), \delta\boldsymbol{oe}(t_{\max})\right) \leq 0 \tag{9.27}$$

where

$$\delta\boldsymbol{\alpha e}(t_{\max}) = \boldsymbol{A}_{c^*}^{-1}(t_{\max})\boldsymbol{x}_{d_i/c^*}(t_{\max}) \tag{9.28}$$

As in the single-agent case, the objective (Equation 9.24) minimizes the change in requested thrust. The first constraint (Equation 9.25) ensures that the thrust does not exceed the maximum thrust the servicer can produce. Noncollision is enforced against the RSO and all other servicers (iterated over as c^*), in the corresponding Hill frame. Constraint (Equation 9.26) ensures that the shielded action does not lead to collision over some upcoming timespan $[t_0, t_{\max}]$, as computed in the preceding lines using the STM for motion relative to c^*. The final constraint (Equation 9.27) uses Equation 9.35 as derived in the next section to maintain long term passive safety on the domain $[t_{\max}, \infty)$, finding the relative orbital elements using the same mapping matrix $\boldsymbol{A}_c^{-1}$ used in the state transition matrix.

9.3.2.2 Conditions for Infinite-Horizon Safety

It is necessary to analyze the long-term trajectories of a deputy relative to a chief in order to provide infinite horizon safety guarantees in the event that the deputy satellite becomes uncontrolled (simply put, satellites should never be on a collision course). Considering the closed-form solution to relative motion with first order eccentricity (see Reference 75, chapter 14.6.3), two cases lead to passive safety: Either the relative motion is periodic, requiring $\delta a = 0$, or some secular term dominates, and the deputy is moving away from the chief.

Only the y component of the motion has such a secular term. The x and z components both oscillate through zero as $t \to \infty$, so they cannot be used to improve the lower bound on the distance between spacecraft. The y component motion is described in terms of orbit element differences

$$y(f, \delta M) \approx a\left(\frac{\delta M}{\eta} + \delta\omega + \cos i\delta\Omega\right) - a\delta_y \sin(f - f_y) - \frac{ae}{2}\sin(2f)\delta e \tag{9.29}$$

where

$$\delta_y(\delta M) = \sqrt{4\delta e^2 + e^2\left(\frac{\delta M}{\eta} - \delta\omega - \cos i\delta\Omega\right)^2} \tag{9.30}$$

By minimizing and maximizing the terms that are periodic in f, the y component motion is bounded

by

$$y(\delta M) \geq \lfloor y(\delta M) \rfloor = a\left(\frac{\delta M}{\eta} + \delta\omega + \cos i\delta\Omega\right) - a\delta_y(\delta M) - \frac{ae}{2}\delta e \tag{9.31}$$

$$y(\delta M) \leq \lceil y(\delta M) \rceil = a\left(\frac{\delta M}{\eta} + \delta\omega + \cos i\delta\Omega\right) + a\delta_y(\delta M) + \frac{ae}{2}\delta e \tag{9.32}$$

which are increasing in δM for $\delta a \neq 0$. If $\delta\dot{M} > 0$ (equivalent to $\delta a < 0$), the deputy is guaranteed to be clear of the chief keepout $\boldsymbol{K}$ for all $t > t^*$ if

$$\lfloor y(\delta M(t^*)) \rfloor \geq (\hat{\boldsymbol{y}}^T \boldsymbol{K} \hat{\boldsymbol{y}})^{-\frac{1}{2}} \tag{9.33}$$

Likewise, if $\delta\dot{M} < 0$ (equivalent to $\delta a > 0$), the condition is

$$\lceil y(\delta M(t^*)) \rceil \leq -(\hat{\boldsymbol{y}}^T \boldsymbol{K} \hat{\boldsymbol{y}})^{-\frac{1}{2}} \tag{9.34}$$

As mentioned earlier, the final case $\delta a = 0$ is always acceptable as it yields periodic motion. These three cases can be combined into two concurrently active constraints, leveraging the increasing nature of the functions so that the cases overlap:

$$\begin{aligned} g_{\text{low}}(\boldsymbol{oe}, \delta\boldsymbol{oe}) &= \delta a\left(\lfloor y \rfloor - (\hat{\boldsymbol{y}}^T \boldsymbol{K} \hat{\boldsymbol{y}})^{-\frac{1}{2}}\right) \leq 0 \\ g_{\text{high}}(\boldsymbol{oe}, \delta\boldsymbol{oe}) &= \delta a\left(\lceil y \rceil + (\hat{\boldsymbol{y}}^T \boldsymbol{K} \hat{\boldsymbol{y}})^{-\frac{1}{2}}\right) \leq 0 \end{aligned} \tag{9.35}$$

9.3.2.3 Iterative Solution Algorithm

The times for which the keepout constraint (Equation 9.26) is enforced must be discretized to make the number of constraints finite. However, if the discretization is too coarse, this introduces the possibility of interstep collisions. While the dynamics used for the single-agent shield allow for a bound on the time discretization to be found that works across all times, such a bound is difficult to compute for these dynamics and results in an excessive number of constraints. To address this, an iterative scheme is used to dynamically determine the time discretization. The position and velocity at each discretized point are used to bound the interstep motion of the deputy.

The minimum distance $R_{d_i/\boldsymbol{K}_{d_i/c^*}}$ between a deputy d_i and the boundary of a keepout region $\boldsymbol{K}_{d_i/c^*}$ is defined as the objective of the optimization problem

$$R_{d_i/\boldsymbol{K}_{d_i/c^*}} = \text{minimize} \quad \|\boldsymbol{r}^* - \boldsymbol{r}_{d_i/c^*}\| \tag{9.36}$$

$$\text{such that} \quad \boldsymbol{r}^{*T} \boldsymbol{K}_{d_i/c^*} \boldsymbol{r}^* \leq 1 \tag{9.37}$$

which is a simple convex problem for nonspherical keepouts and trivially

$$R_{d_i/\boldsymbol{K}_{d_i/c^*}} = \|\boldsymbol{r}_{d_i/c^*}\| - R_{d_i/c^*} \tag{9.38}$$

for spherical keepouts with radius R_{d_i/c^*}. It follows that the worst-case maximum timestep between t_i and t_{i+1} that satisfies the keepout with a safety margin of ϵ is given by

$$t_{i+1} - t_i \leq \frac{R_{d_i/\boldsymbol{K}_{d_i/c^*}}(t_i) + \epsilon}{\max\limits_{t \in [t_i, t_{i+1}]} \|\boldsymbol{v}_{d_i/c^*}(t)\|} \tag{9.39}$$

Because this condition contains a continuous term, the condition is approximated using the maximum of the endpoints of the segment

$$t_{i+1} - t_i \leq \min\left(\frac{R_{d_i/\boldsymbol{K}_{d_i/c^*}}(t_i) + \epsilon}{\max\left(\|\boldsymbol{v}_{d_i/c^*}(t_i)\|, \|\boldsymbol{v}_{d_i/c^*}(t_{i+1})\|\right)}, \Delta t_{\max} \right) \tag{9.40}$$

This approximation is exact when $\boldsymbol{v}$ is monotonic on $[t_i, t_{i+1}]$ and is otherwise reasonable due to the slow speed of orbital dynamics, as long as some $\Delta t_{\max}$ is selected that is small relative to the dynamics. Additionally, since the timestep must be smaller in higher-risk situations with a higher velocity or lower distance, the period covered by the step is more likely to be monotonic and thus the approximation exact.

The method solves the optimization problem, updating the time discretization each iteration until Equation 9.40 is satisfied at all sample points. The complete method is given in Algorithm 13 and Algorithm 14. First, the sample time vector is seeded uniformly. Then, the optimization problem is solved to find the shielded action. If the algorithm has converged (i.e. Equation 9.40 is satisfied at all times), the algorithm has converged and the solution is returned. If not, the time discretization is resampled based on the solution trajectory.

To resample the time discretization $\boldsymbol{t}'$ from the previous discretization $\boldsymbol{t}$, the necessary discretization at the current steps is approximated as a continuous function with a linear interpolation of Equation 9.40 at the sample points. Starting at t_0, points are added to the sample vector such that the Δt between the new point and the previous point is less than $(1 - \kappa)$ times the lowest

Algorithm 13 Iterative Shield Solution

$\boldsymbol{t}^{(c^*)} \leftarrow [t_0 : \Delta t_{\max} : t_{\max}]\ \forall c^* \in \{c, d_1, ..., d_N | c^* \neq d_i\}$
while not converged **do**
    solution $\leftarrow$ solve Equation 9.24-9.27
    if Equation 9.40 $\forall\ t_i \in \boldsymbol{t}\ \forall c^* \in \{c, d_1, ..., d_N | c^* \neq d_i\}$ **then**
        return solution
    else
        for $c^* \in \{c, d_1, ..., d_N | c^* \neq d_i\}$ **do**
            $\boldsymbol{t}^{(c^*)} \leftarrow$ ResampleTimes(solution$^{(c^*)}$)
        end for
    end if
end while

interpolated maximum allowable value of Δt in that period, where $0 < \kappa < 1$. The purpose of retaining all old sample points when resampling the time discretization in Algorithm 14 is to prevent chatter between two different classes of solutions with different discretizations. By retaining the old points, the constraints that lead to previous iterations' solutions are always maintained.

9.3.2.4 Argument for Constellation-Wide Safety

The combination of asynchronous decision-making and the shield's safety guarantees ensure relative safety for all agents across all time. Assuming that all agents start in passively safe relative trajectories, when a single satellite executes a new control, it needs only to ensure its own passive safety relative to all other satellites in order to maintain constellation-wide safety. Agents are unlikely to want to retask at the same time [27], and if they did, one could simply be prioritized and tasking could precede sequentially. As with the single-agent shield, the trivial solution $\delta\boldsymbol{v} = -\Delta\boldsymbol{v}$ always exists. As a result, the size of the shield's optimization problem only grows linearly in constraints with the number of servicers, as opposed to a scheme in which all satellites compute new controls at the same time, which would cause quadratic growth in the optimization problem size.

Algorithm 14 Resample Times

function ResampleTimes(solution, $\kappa = 0.1$)
  $\boldsymbol{t}' \leftarrow [t_0]$
  $\Delta t_{\text{interp}}(t) \leftarrow$ linear interpolation of R.H.S. of Equation 9.40
  if $\boldsymbol{t}'_{-1} \geq t_0 + t_{\max}$ **then**
    $\boldsymbol{t}'_{-1} = t_0 + t_{\max}$
    return $\boldsymbol{t}'$
  else
    $t_i = \boldsymbol{t}'_{-1}$
    $t_{\text{new}} \leftarrow t_{\text{new}} - t_i = (1 - \kappa) \min_{t \in [t_i, t_{\text{new}}]} \Delta t_{\text{interp}}(t)$
    if $\exists\ t_{\text{old}} \in \boldsymbol{t}$ s.t. $t_i < t_{\text{old}} < t_{\text{new}}$ **then**
      append$(\boldsymbol{t}', t_{\text{old}})$
    else
      append$(\boldsymbol{t}', t_{\text{new}})$
    end if
  end if
end function

9.4 Single-Agent Inspection

Policies are trained and benchmarked for the RL formulation across a range of fuel use parameters α. The impact of the eccentric-chief shield is validated and evaluated across the benchmarks between the unshielded and shielded cases. The fuel-time Pareto front across values of α are compared for the unshielded and shielded policies. Individual trajectories are also inspected, showing the difference between the unshielded RL trajectory and the shielded trajectories with varying keepout radius. These policies are referred to as $\pi_{\text{single},\alpha}$.

9.4.1 Training Agents for Fuel and Time Efficiency

Policies $\pi_{\text{single},\alpha}$ are trained in the high-fidelity implementation of this environment, without a shield, for a range of fuel use penalties $\alpha \in \{0.01, 0.05, 0.10, 0.20, 0.30\}$[1] . The RLlib implementation of proximal policy optimization (PPO) is used to train policies [34, 129]. The algorithm and training pipeline are modified to use semi-Markov discounting for advantage estimation [26]. The policies are represented by a fully connected multilayer perceptron (MLP) NN of size 2×256, which takes

[1] The process of training the $\alpha = 0.1$ policy was interrupted at 25.4M steps, but the policy is still analyzed since training has converged.

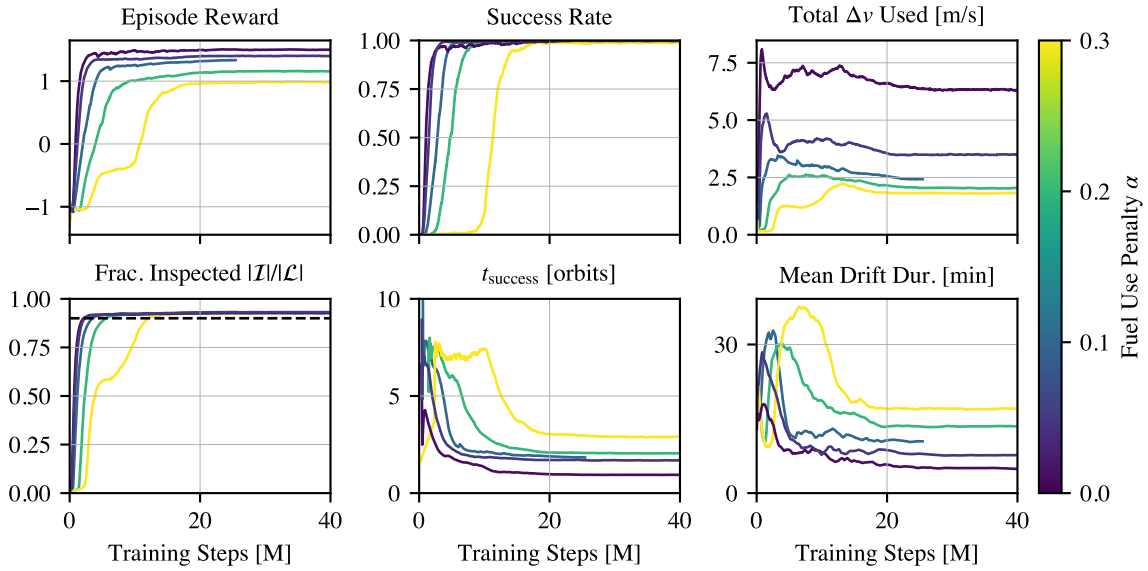


Figure 9.2: Training curves for $\pi_{\text{single},\alpha}$ policies in an $N = 1$ servicer environment with varying fuel use penalty α.

about 100 μs per decision to evaluate in inference. The time use penalty is $\alpha_t = 0.05$/orbit, and $\beta_{\text{success}} = \beta_{\text{fail}} = 1.0$. The policies presented in this chapter are a refined version of those presented in Reference 98, with a 40M-step learning rate schedule of 10M steps at 10^{-4}, 10M decreasing to 10^{-5}, 10M decreasing to 10^{-6}, and 10M holding at 10^{-6}. A discount factor of $\gamma = 0.9999$ is used. Hyperparameter values are selected following an ablation study across algorithm and network parameters.

Figure 9.2 shows how various metrics evolve as the policies learn to complete the task. All the policies reach a success rate of nearly 100% as they learn to inspect at least 90% of illuminated points. The fuel and time used are both driven down as training refines the policies. As expected, policies for lower α are faster (lower t_{success}) but less fuel-efficient (higher total Δv used) when completing the task. The policies tend to prefer relatively short drifts between a high number of small impulsive thrusts.

9.4.2 Unshielded & Shielded Performance

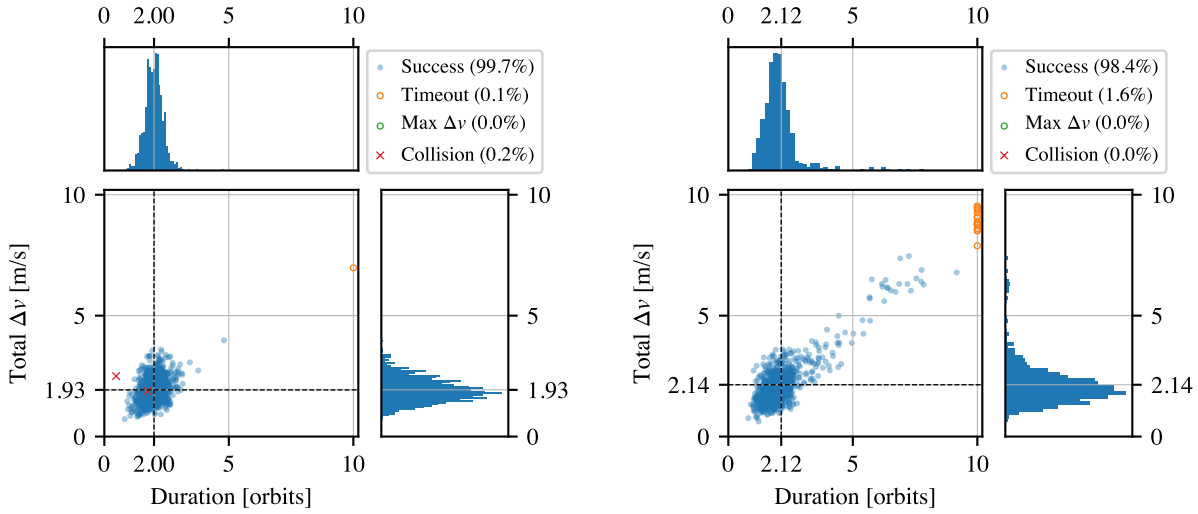
The policy $\pi_{\text{single},\alpha=0.20}$ is evaluated with and without the shield for 1000 environment seeds to investigate the impact of the eccentric-chief shield on policy performance. The shield uses a 5-meter keepout, $t_{\max} - t_0 = 4$ orbits, and $\Delta t_{\max} = 500$ seconds.

The unshielded (Figure 9.3a) and shielded (Figure 9.3b) performance of the policy $\pi_{\text{single},\alpha=0.20}$ is compared. On average, the shield results in trajectories that take 0.12 more orbits and 0.21 m/s more fuel to complete the inspection task. As expected, the shield eliminates the (relatively rare) collisions produced by the unshielded policy. However, this comes at the cost of increased time—and thus fuel—to complete the task. Practically, the cases where the shielded policy does not complete the task are not critically bad: an inspection threshold of $> 80\%$, rather than the 90% threshold, is met in a reasonable time. Another reason for the shield's strong impact on performance is that it is not **only** preventing collisions—it also introduces infinite-horizon passive safety guarantees that are stricter than the base problem's non-collision constraint.

Figure 9.4a gives histograms of the size of shield interventions and Figure 9.4b shows how long the shield took to compute on an Apple M2 Pro CPU for the cases in Figure 9.3b. In the 33.8% of actions where the shield activates, the shield yields an average $\|\delta \boldsymbol{v}\|$ of only 0.04 m/s, showing that it is generally unobtrusive. The evaluation time is minimal, making the shield compatible with the closed-loop paradigm of the RL-based policy.

The histograms in Figure 9.3 for the policy $\pi_{\text{single},\alpha=0.20}$ can be compared for policies across the range of trained α. The results are shown in Figure 9.5. Ultimately, the shield leads to a relatively small performance degradation on average, sometimes even marginally improving performance. However, across all values of α the variance in performance grows considerably when the shield is enabled. This effect is especially strong for cases that are more heavily biased towards fuel or time optimality, as the resulting unshielded trajectories may be "riskier" to achieve such performance extremes.

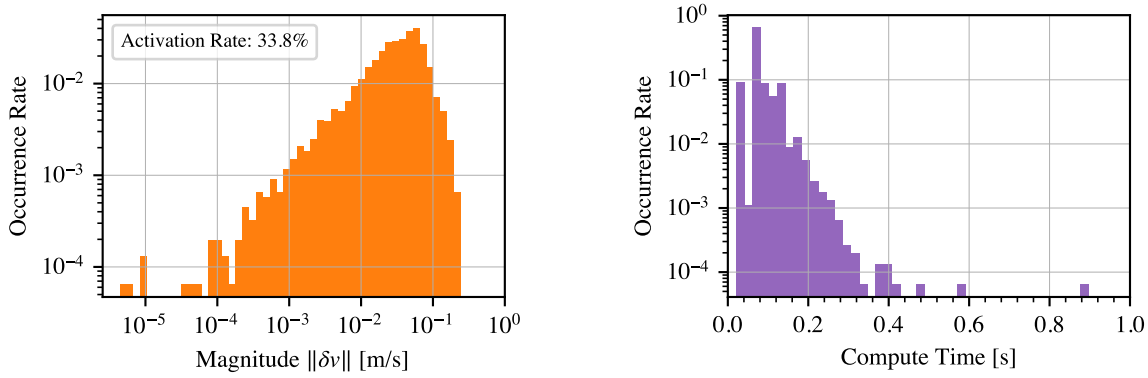
For an absolute sense of optimality, see Figure 10.3, which compares the unshielded poli-



(a) Unshielded performance.

(b) Shielded performance.

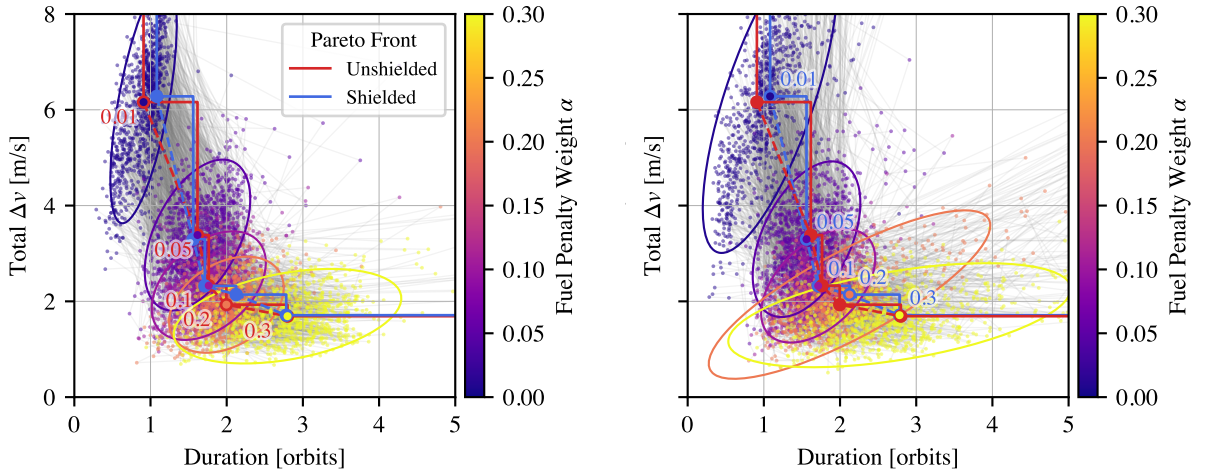
Figure 9.3: Performance of the policy $\pi_{\text{single},\alpha=0.20}$ in the environment with $N = 1$ servicer across 1000 trials.



(a) Distribution of shield magnitudes.

(b) Distribution of time to compute shield.

Figure 9.4: Shield statistics for cases in Figure 9.3b.



(a) Unshielded performance.

(b) Shielded performance.

Figure 9.5: Mean Pareto fronts across 1000 trials for unshielded and shielded single-agent policies $\pi_{\text{single},\alpha}$; 2σ covariances.

cies' Pareto front against the probabilistic roadmap (PRM)-based approach to trajectory planning developed in chapter 10.

9.4.3 Example Trajectories

Trajectories are generated for a high illumination case (Figure 9.6) and a low illumination case (Figure 9.7) for the policy $\pi_{\text{single},\alpha=0.10}$, both unshielded and with a range of shield radii; this highlights the ability of the shield to enforce a different keepout radius than the policy was trained for. Both the Hill-frame trajectory and various statistics (Δv used, fraction of points inspected, and RSO-servicer distance) are recorded.

In both cases, there is an initial high-cost burn to near the RSO. Once within the inspection radius, points are inspected at a relatively constant rate, other than when the spacecraft is in eclipse (shaded gray). The geometry of trajectories are similar with or without shields: In the high illumination case, the servicer tracks the illuminated side of the RSO as it moves, while in the low illumination case, only one side of the RSO is ever illuminated, so the servicer stays on that side.

The shield is seen to successfully enforce the various keepout radii, as denoted by dashed lines

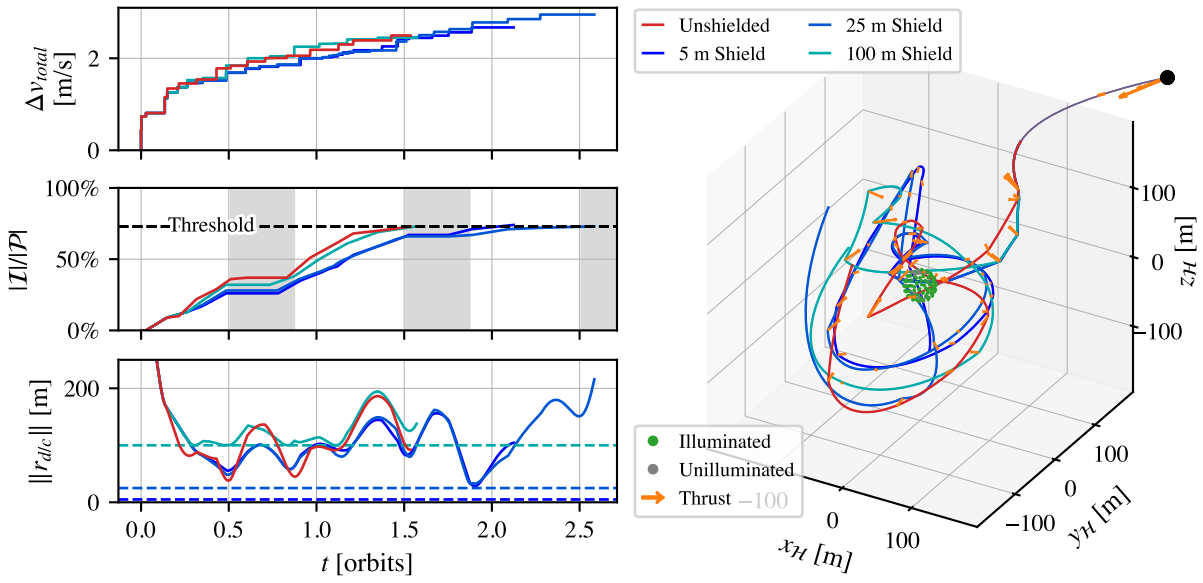


Figure 9.6: Unshielded and shielded trajectories generated by the policy $\pi_{\text{single},\alpha=0.10}$ in an environment with $N = 1$ servicer, $\beta = 7.6°$.

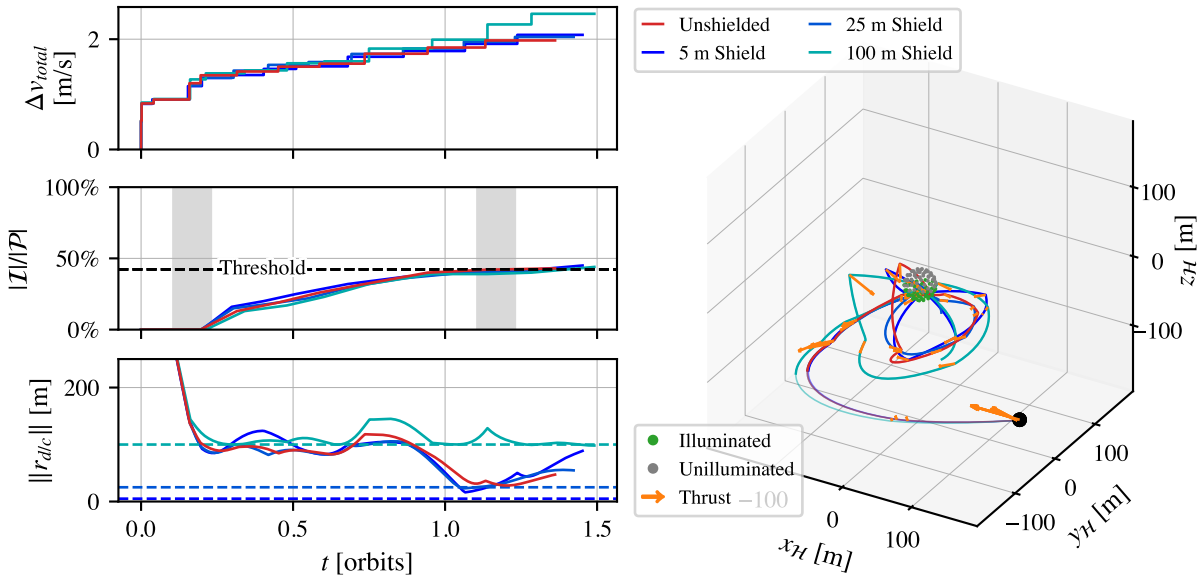


Figure 9.7: Unshielded and shielded trajectories generated by the policy $\pi_{\text{single},\alpha=0.10}$ in an environment with $N = 1$ servicer, $\beta = -62.9°$.

in the plot of $\|\boldsymbol{r}_{d/c}\|$. While the difference from the unshielded trajectory is generally minimal, the 100 m keepout leads to notable differences in the trajectory, with the shield keeping the servicer at the boundary of the constraint much of the time.

9.5 Multi-Agent Inspection

A multi-agent formulation of the problem with $N = 2$ servicers is considered. In the first approach, the single-agent policy $\pi_{\text{single},\alpha}$ is deployed in the multi-agent environment, using the closed-loop response to the agents' joint observation of inspection level to attempt to induce collaboration. In the second approach, new policies $\pi_{\text{multi},\alpha}$ are trained in the multi-agent environment so that they actively learn to work together. Again, these policies are trained and benchmarked across a range of fuel use penalties α, both with and without the shield.

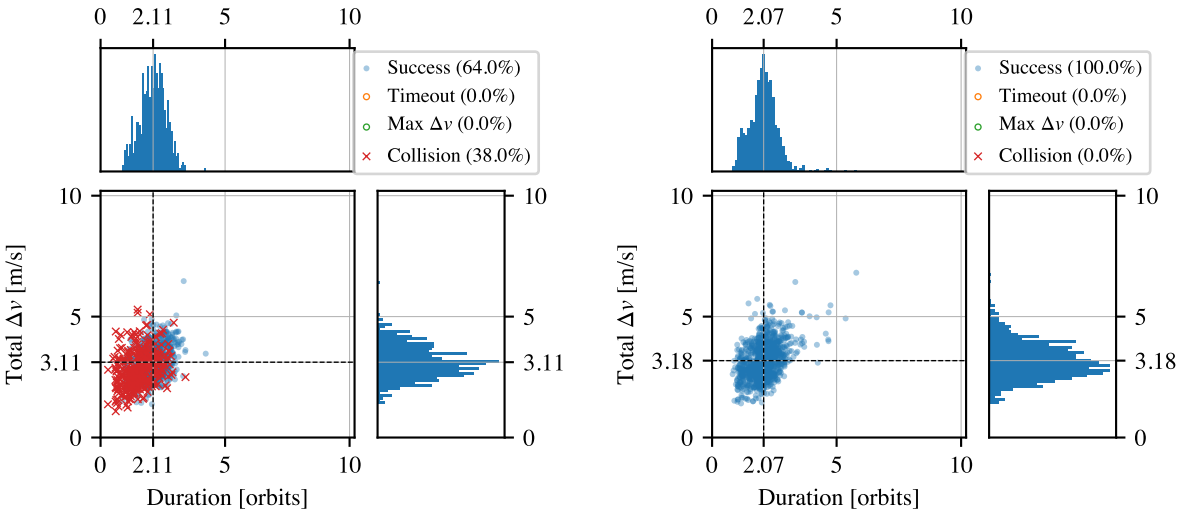
9.5.1 Single-Agent Policy in Multi-Agent Deployment

A first attempt at multi-agent inspection is inspired by the approach taken in chapter 6, in which multiple Earth-observing satellites (EOSs) using policies trained in a single-agent environment were successfully deployed in a multi-agent environment because of their closed-loop response to the environment state. In that chapter, cooperation was induced via a shared task completion list. Similarly, two servicers execute a single-agent policy $\pi_{\text{single},\alpha}$ while sharing and responding to their observation of the joint inspection status of the RSO.[2]

This method is benchmarked across 1000 cases with and without the shield in Figure 9.8a for the policy $\pi_{\text{single},\alpha=0.30}$. Compared to the single-agent case (Figure 9.3), the impact of the shield is minimal both in terms of mean performance and distribution shape.

The single-agent-policy-in-multi-agent-environment benchmark is the strongest test of the shield. The shield successfully eliminates very frequent collisions between the two servicers in all cases, even though each servicer has no knowledge of the other's existence or intrinsic motivation

[2] Critically different from the methods presented in chapter 6, no future intended completion status is shared among agents due to the high computational cost of finding what RSO points may be inspected on the forward-propagated trajectory.



(a) Unshielded performance.

(b) Shielded performance.

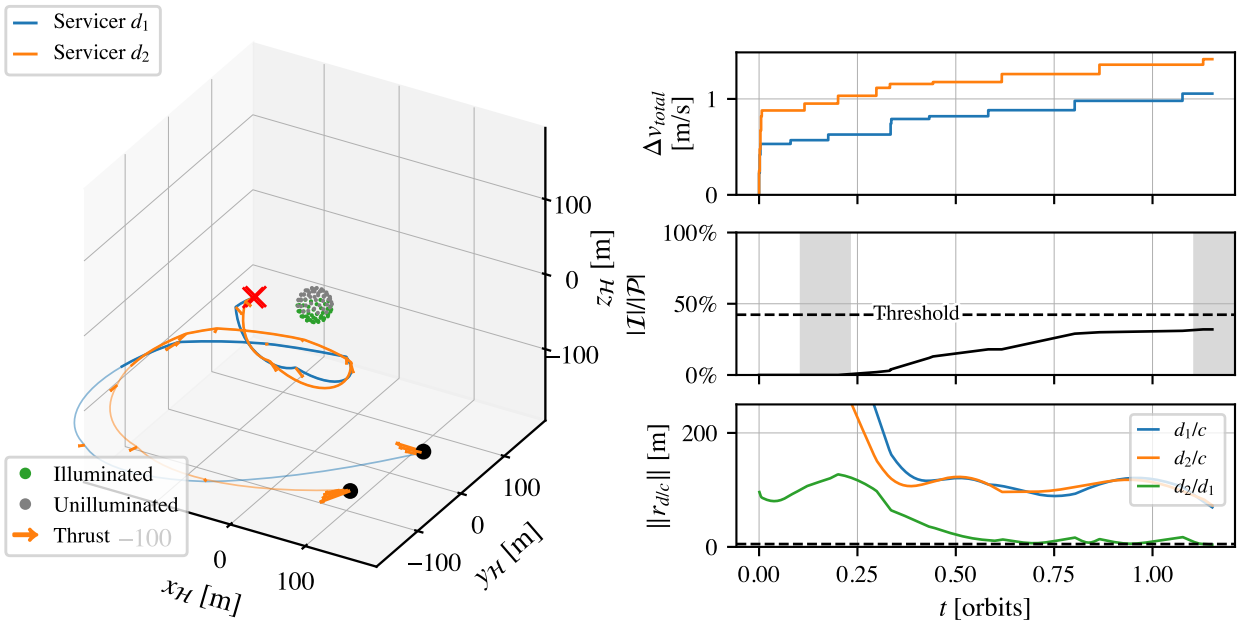
Figure 9.8: Performance of the policy $\pi_{\text{single},\alpha=0.30}$ in the environment with $N = 2$ servicers across 1000 trials.

to avoid collision. The shield eliminates all collisions, which have a prevalence of 38.0% to 87.2% depending on the unshielded policy (Table 9.3). For this architecture, the shield is essential.

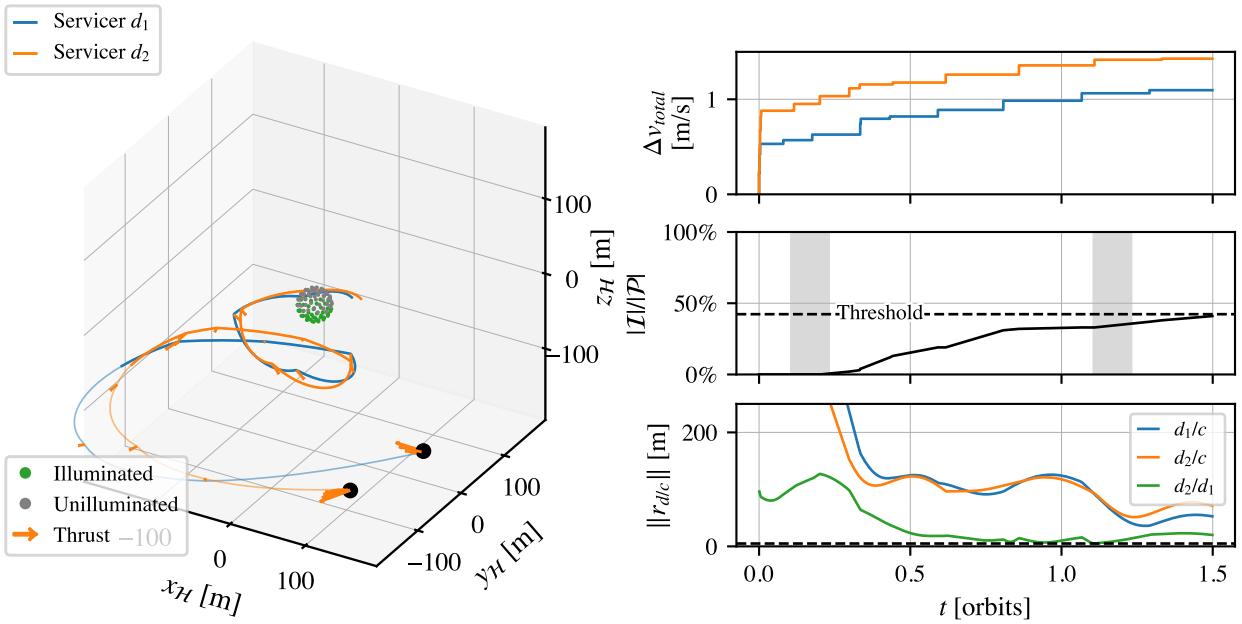
Undesirable behavior is frequently seen when using this method for multi-agent inspection, as observed in the example trajectories given in Figure 9.9 and Figure 9.10. The two servicers tend to converge to the same trajectory for inspection, as one servicer does not anticipate another servicer's activities. This leads to the high number of collisions observed in the unshielded benchmarks, since both agents want to be in the same location at the same time. While the shield can prevent collisions, it does not diversify the servicer trajectories enough to improve the effectiveness of the agents at completing the task; the shielded trajectories of the two agents are still similar. In some cases (such as in Figure 9.10b), the shield induces a behavior in which the two servicers "fight" to be on the same trajectory, leading to high inspection durations and fuel costs.

9.5.2 Multi-Agent Training

The undesirable trajectory matching behavior of the policy $\pi_{\text{single},\alpha}$ in the multi-agent setting motivates the need for policies trained in the multi-agent environment in order to actively diversify

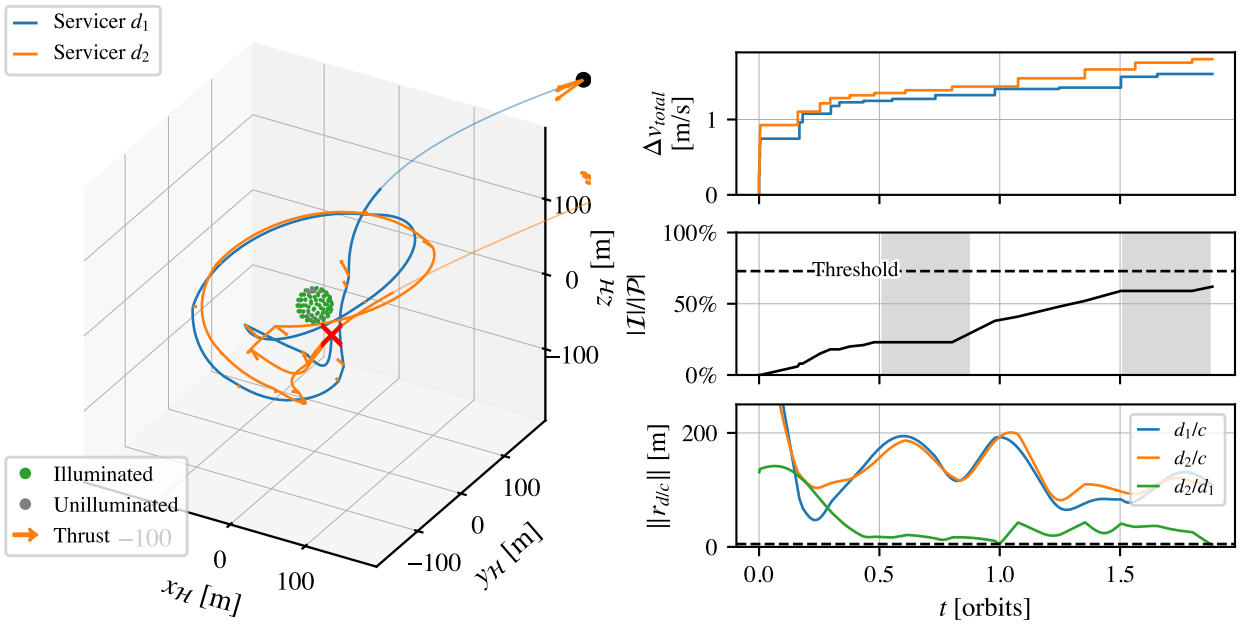


(a) Unshielded trajectories, resulting in collision.

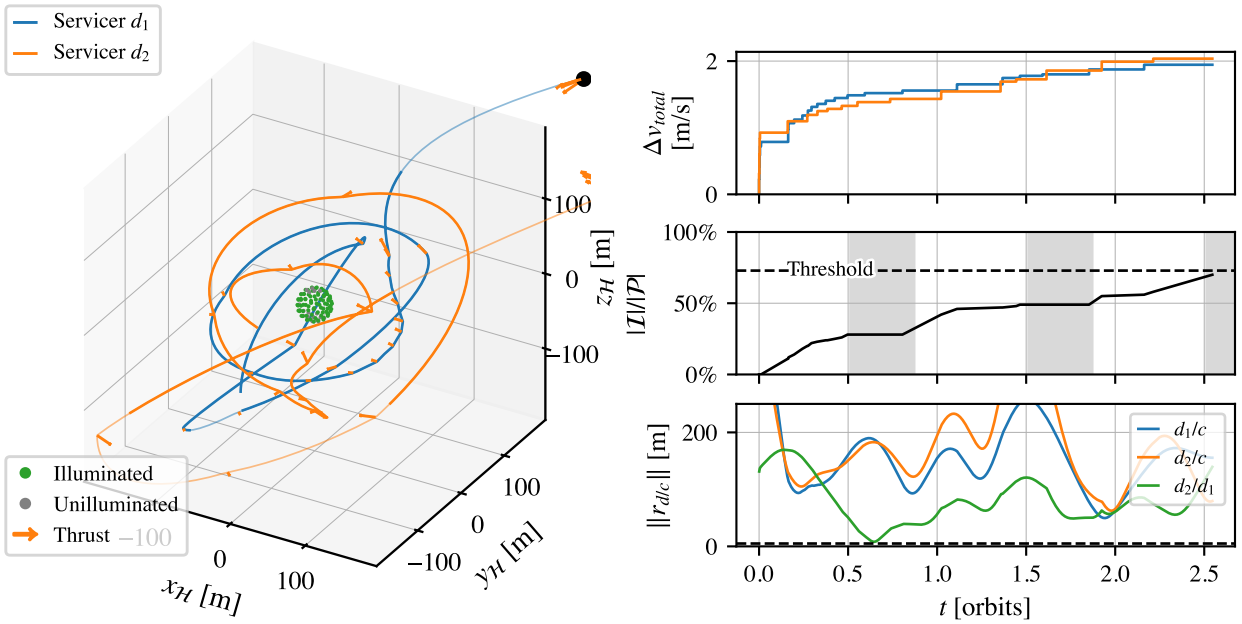


(b) Shielded trajectories.

Figure 9.9: Trajectories generated by the policy $\pi_{\text{single},\alpha=0.30}$ in an environment with $N = 2$ servicers, $\beta = 7.6°$.



(a) Unshielded trajectories, resulting in collision.



(b) Shielded trajectories.

Figure 9.10: Trajectories generated by the policy $\pi_{\text{single},\alpha=0.30}$ in an environment with $N = 2$ servicers, $\beta = -62.9°$.

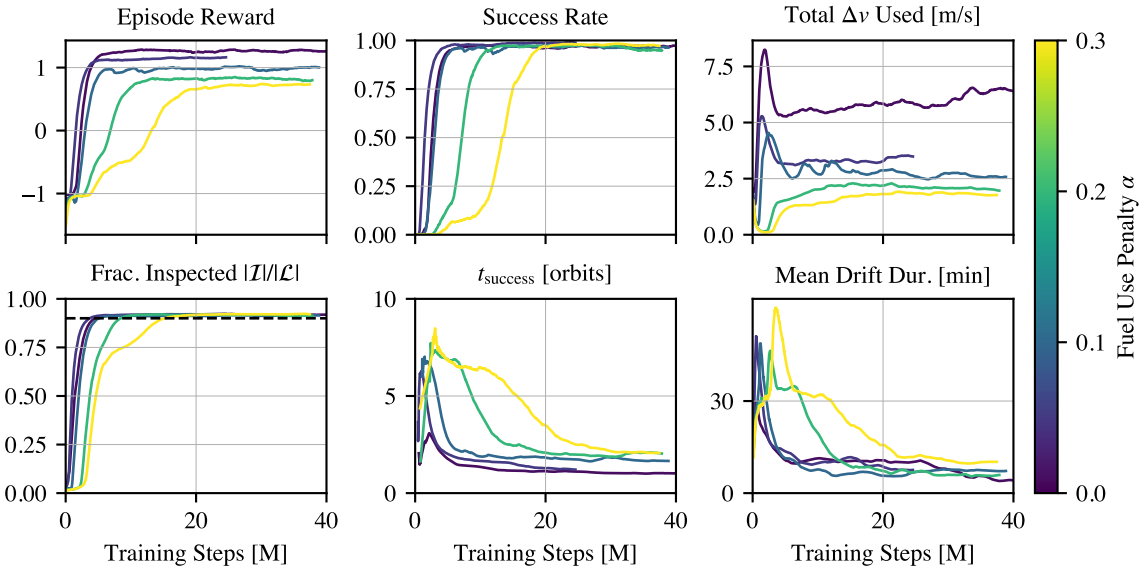


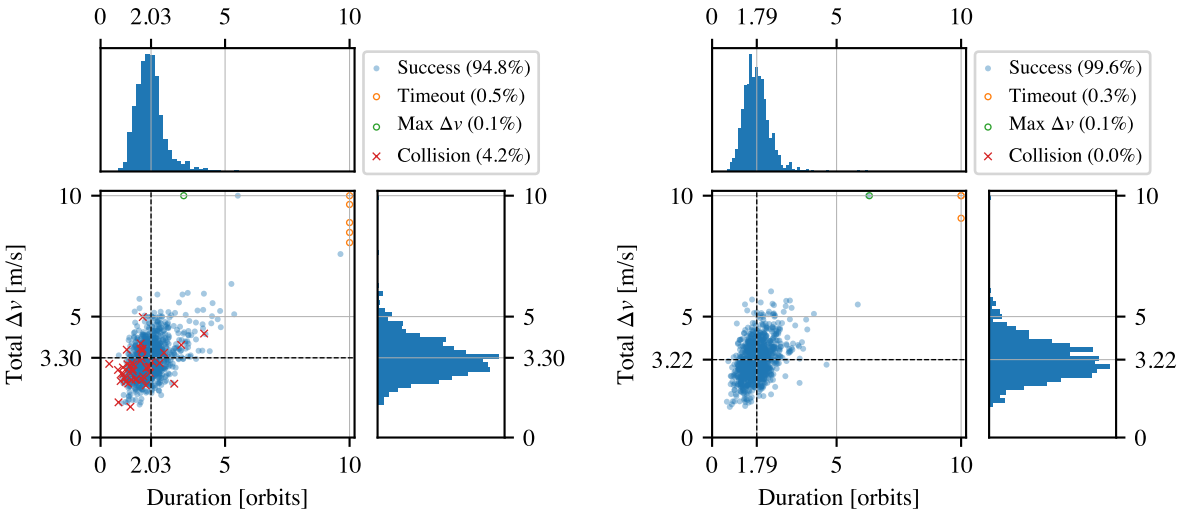
Figure 9.11: Training curves for $\pi_{\text{multi},\alpha}$ policies in an $N = 2$ servicer environment with varying fuel use penalty α.

the inspection trajectories of different servicers. PPO is used with the necessary modifications for asynchronicity and sMDPs; similar to the multi-agent architecture described in chapter 7, all agents in the environment share copies of the same policy and all experience tuples from those agents contribute to training for the policy. Here, the additional observations in Table 9.2 are provided to the agents. The same hyperparameters are used as in single-agent training for this problem. The resulting policies are referred to as $\pi_{\text{multi},\alpha}$.

In Figure 9.11, the performance of the multi-agent training pipeline is shown. Training converged in a reasonable time for policies $\pi_{\text{multi},\alpha}$ with $\alpha \in \{0.01, 0.05, 0.10, 0.20, 0.30\}$, the same values trained for $\pi_{\text{single},\alpha}$. Overall, the trends are comparable to those seen in single-agent training (Figure 9.2).

9.5.3 Multi-Agent Policy Deployment

Again, a benchmark is performed over 1000 trials for the unshielded and shielded policy $\pi_{\text{multi},\alpha=0.30}$; the results are given in Figure 9.12a and Figure 9.12b, respectively. Compared to the unshielded and shielded benchmarks of $\pi_{\text{single},\alpha=0.30}$, there is significantly less deformation



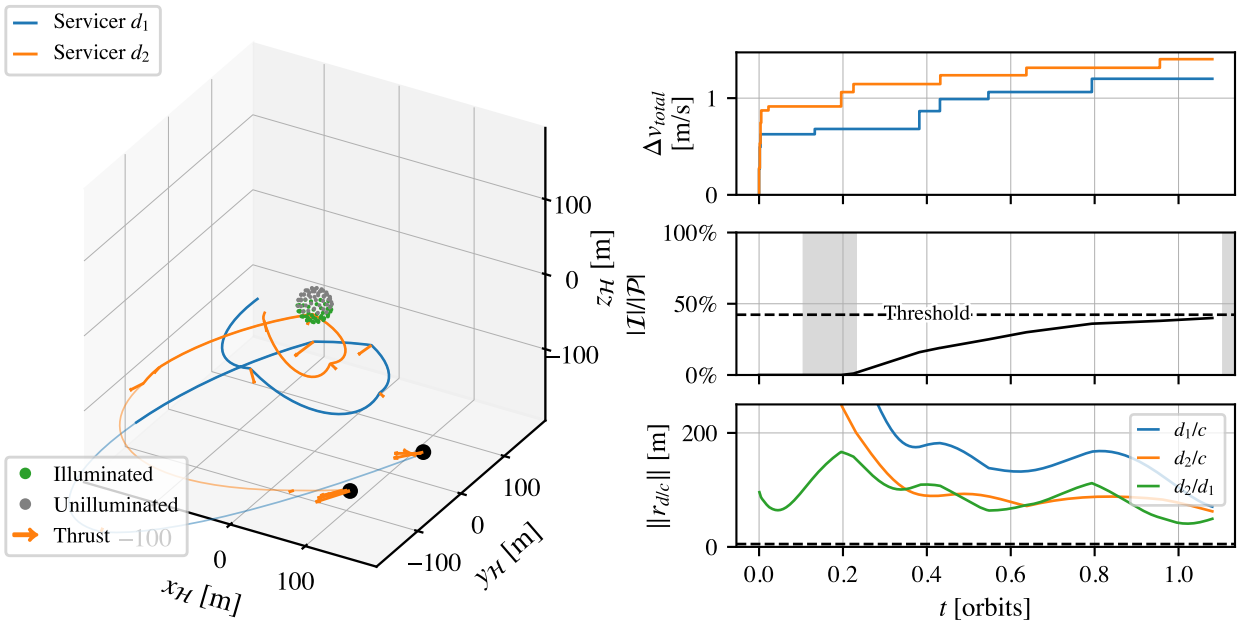
(a) Unshielded performance.

(b) Shielded performance.

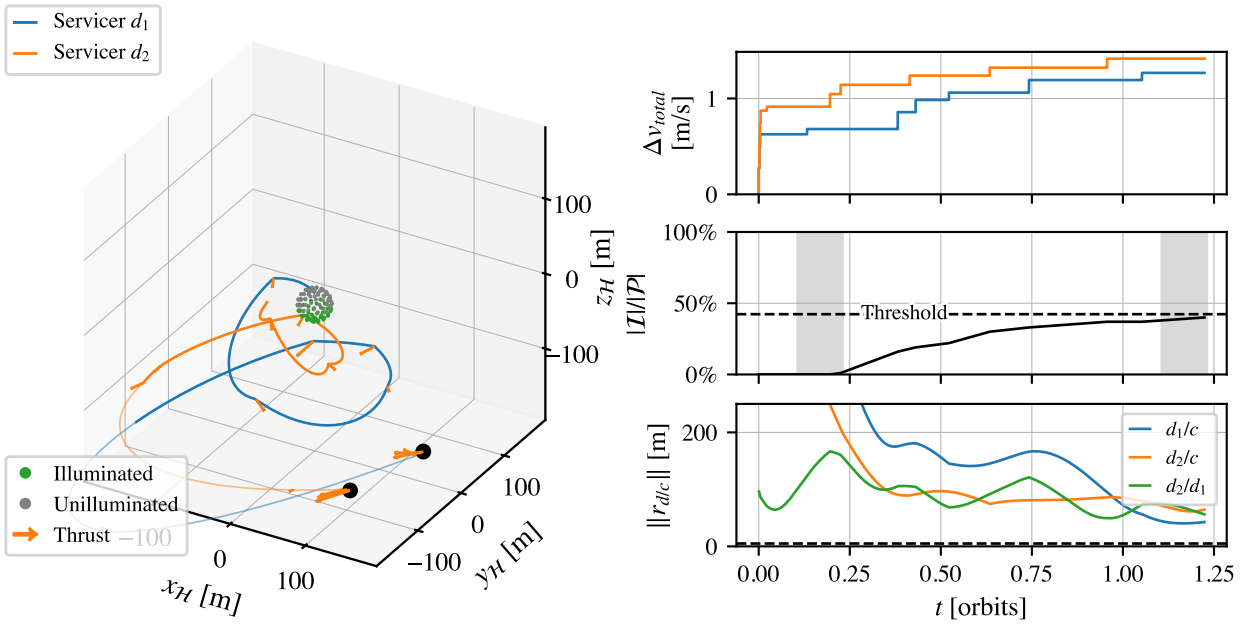
Figure 9.12: Performance of the policy $\pi_{\text{multi},\alpha=0.30}$ in the environment with $N = 2$ servicers across 1000 trials.

between the two histograms, implying that the actions taken by $\pi_{\text{multi},\alpha=0.30}$ are safer and need less intervention. For this policy, the shield actually increases performance. Additionally, because the policies $\pi_{\text{multi},\alpha}$ are penalized for collision between servicers in training, the fraction of unshielded cases ending in collision is relatively low. As Table 9.3 reflects, this improvement is true across values of α.

Figure 9.13 and Figure 9.14 compare the unshielded and shielded performance of policy $\pi_{\text{multi},\alpha=0.30}$ with $N = 2$ servicers for the same cases as previous example trajectories were evaluated on. While the unshielded case in Figure 9.13a does exhibit a collision between the servicers, these are relatively rare as reflected in Table 9.3. Across the cases, the two agents travel on significantly different trajectories, often with one inspecting while the other is out of inspection range on a larger passive maneuver. This demonstrates that the other-servicer-aware training scheme leads to a diversification of trajectories between cooperative servicers, as intended.

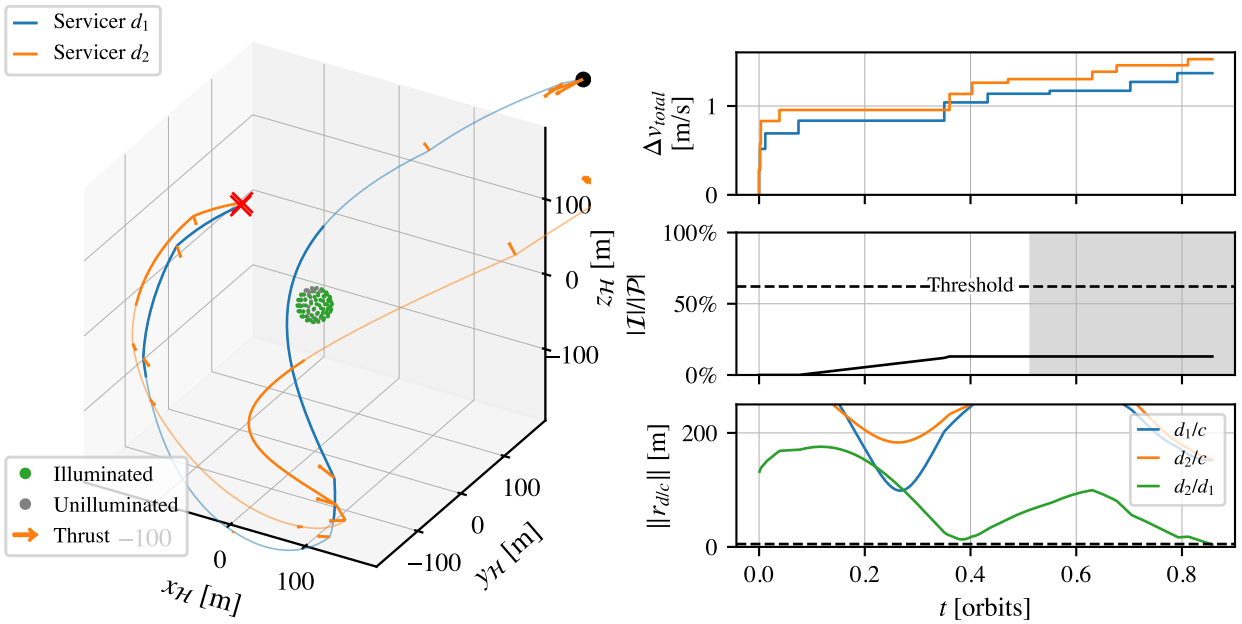


(a) Unshielded trajectories, not resulting in collision.

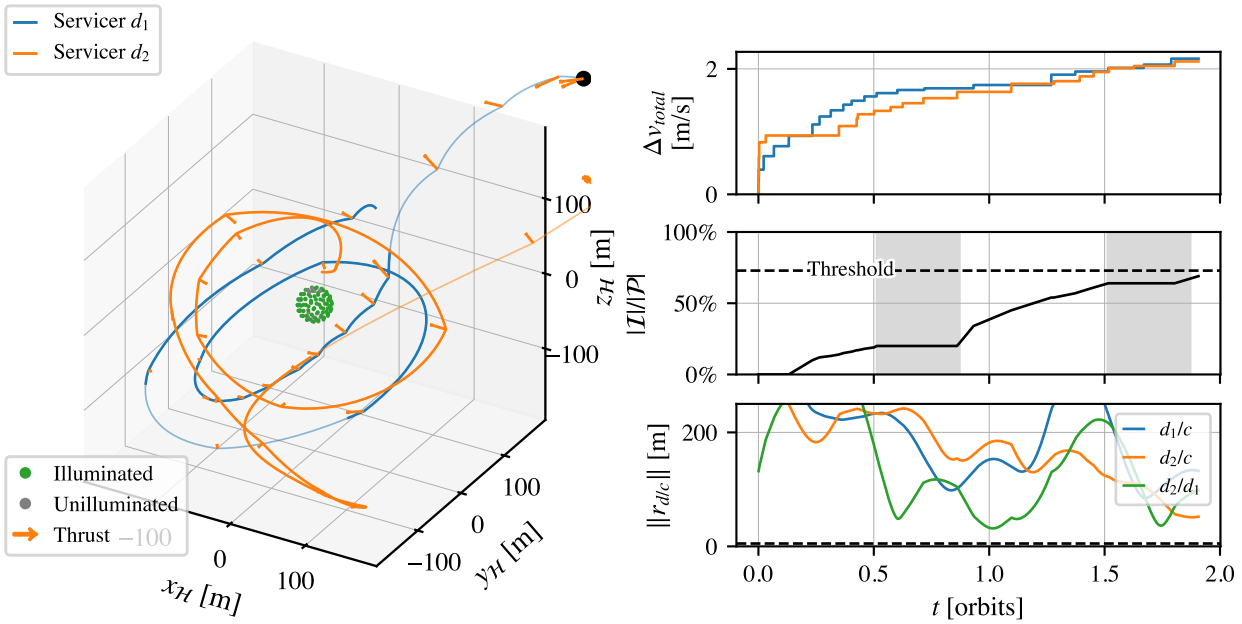


(b) Shielded trajectories.

Figure 9.13: Trajectories generated by the policy $\pi_{\text{multi},\alpha=0.30}$ in an environment with $N = 2$ servicers, $\beta = 7.6°$.



(a) Unshielded trajectories, resulting in collision.



(b) Shielded trajectories.

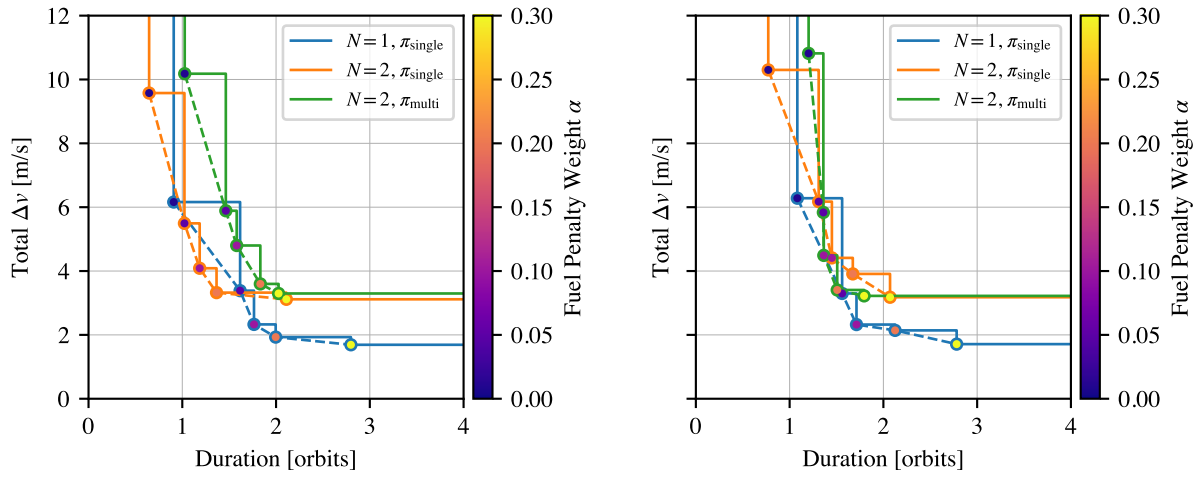
Figure 9.14: Trajectories generated by the policy $\pi_{\text{multi},\alpha=0.30}$ in an environment with $N = 2$ servicers, $\beta = -62.9°$.

9.5.4 Comparing Multi-Agent Inspection Methods

Finally, the single-agent and two multi-agent reinforcement learning (MARL) inspection methods are compared across the fuel-time Pareto front parameterized by α. Two questions are investigated: How does the policy $\pi_{\text{single},\alpha}$ compare to the policy $\pi_{\text{multi},\alpha}$ in the $N = 2$ servicer environment, and does having two servicers benefit performance at all for this task?

Table 9.3: Unshielded policy collision rate in $N = 2$ servicer deployment environment, 1000 trials.

Fuel Penalty α	**0.01**	**0.05**	**0.10**	**0.20**	**0.30**
Policy $\pi_{\text{single},\alpha}$	87.2%	89.0%	73.7%	85.1%	38.0%
Policy $\pi_{\text{multi},\alpha}$	1.1%	7.9%	3.6%	11.1%	4.2%



(a) Unshielded performance, non-colliding cases only.

(b) Shielded performance.

Figure 9.15: Mean Pareto fronts across 1000 trials for single- and multi-agent inspection policies.

Figure 9.15 compares fuel-time Pareto fronts across the three combinations of servicer counts and policies, both for unshielded and shielded configurations. For cases with the $N = 2$ servicer environment, the sum of both servicers' Δv is reported. In the unshielded case (Figure 9.15a), the front is dominated by $\pi_{\text{single},\alpha}$ with $N = 1$ for low-fuel trajectories and $N = 2$ for low time trajectories. However, only non-colliding cases are reported on the Pareto front; because that policy has a high collision rate for $N = 2$ as listed in Table 9.3, the performance is misleading.

The shielded Pareto fronts (Figure 9.15b) provide a better comparison of the true relative performance. Here, the performance of the multi-agent-aware policy $\pi_{\text{multi},\alpha}$ largely dominates the front for the single agent policy $\pi_{\text{single},\alpha}$ in the $N = 2$ environment. Qualitatively, the diverse trajectories producing different viewing aspects are likely more beneficial to users; however, this is not reflected numerically in the reward model for this formulation.

Ultimately, the $N = 1$ environment dominates the majority of the front. This implies that having multiple servicers is at best not beneficial in the case of a small spherical RSO and often worse. Once accounting for the additional costs of getting multiple servicers in range of the RSO, there is little justification for having multiple servicers in this environment.

With the small region of the spherical RSO that is illuminated at any given time, the size of the region of inspectability is the limiting factor for the rate of inspection, not the number of servicers. More servicers lead to lazy or duplicative agents that add additional collision safety constraints to the trajectories. In future work, this problem should be considered for larger and more complex RSOs that could benefit from multi-agent inspection.

Chapter 10

Planning for Inspection with Probabilistic Roadmaps

10.1 Motivation

Just as the mixed-integer linear program (MILP) solver provides a baseline for comparison against reinforcement learning (RL) in the agile Earth-observing satellite scheduling problem (AEOSSP), the performance of the RL-based policies in the resident space object (RSO) inspection problem can only be properly judged against a pseudo-optimal solver. This chapter develops an alternative solution to the inspection problem that utilizes probabilistic roadmaps (PRMs) and mixed-integer quadratically-constrained quadratic programs (MIQCQPs) to find solutions for the RSO inspection problem defined in chapter 8.

Beyond RL, two classes of methods are reasonably considered for solving the RSO inspection problem: non-RL-based Markov decision process (MDP) solution techniques, and sample-based motion planning. For the former, the continuous action space makes typical sequential problem solvers like Monte Carlo tree search (MCTS) and variants thereof perform very poorly; the high dimensionality of the state space further complicates tree search. Because the structure of the problem is well known, motion planning techniques that exploit knowledge of the dynamics and objectives are more appealing.

Sample-based motion planners offer a general framework for trajectory planning and optimization. Algorithms building on rapidly-exploring random tree (RRT), such as stable sparse RRT (SST) [167], among many others, can solve kinodynamically-constrained path planning problems across domains. Considering rendezvous and proximity operations (RPO) tasks specifically, tra-

ditional A*-based path planning methods are combined with methods for robustness to thruster failures in References 168 and 169. However, for the inspection task, the goal state of fully observing the RSO makes the problem requirements considerably more complex than point-to-point motion planning. To apply algorithms like SST directly, a high-dimensional motion-planning state must be used that includes a representation of RSO facet observation. The resulting problem is thus too high-dimensional for tree-based exploration to solve efficiently.

In this chapter, the RSO inspection problem is solved for a single satellite with an algorithm that iteratively builds a PRM and finds an objective-satisfying trajectory through it with a MIQCQP. The performance of the algorithm is compared with the RL-based policies over the fuel-time Pareto front.

10.2 The PRM-Based Algorithm

The PRM-based approach for the problem consists of iteratively constructing a roadmap of waypoints, then using a MIQCQP to find a trajectory through the roadmap that satisfies the task constraints.

10.2.1 Algorithm Structure

Algorithm 15, PlanInspectionTrajectory, gives the general scheme of the trajectory planning algorithm. The trajectory space is discretized into waypoints $w \in \mathcal{W}$ that consist of the RSO Hill- and body-frame position of the servicer and the mean anomaly of the waypoint, $w_w = (\boldsymbol{r}_w, M_w)$. In the first iteration, these waypoints are sampled in regions of the state space that have inspectable RSO points in $\mathcal{L}$ (SeedWaypoints). In following iterations, the waypoints are resampled around the incumbent solution from the previous iteration in order to improve beyond the initial discretization (ResampleWaypoints). With the waypoints sampled, a roadmap of dynamically feasible transitions is built between waypoints (ConnectGraph). The roadmap is then encoded in a MIQCQP that optimizes for fuel and time (weighted by ζ, which is akin to α in the MDP formulation) with constraints for feasibility and complete inspection. The solution to

the MIQCQP provides a trajectory that satisfies the inspection task requirements.

Algorithm 15 Inspection trajectory planning function

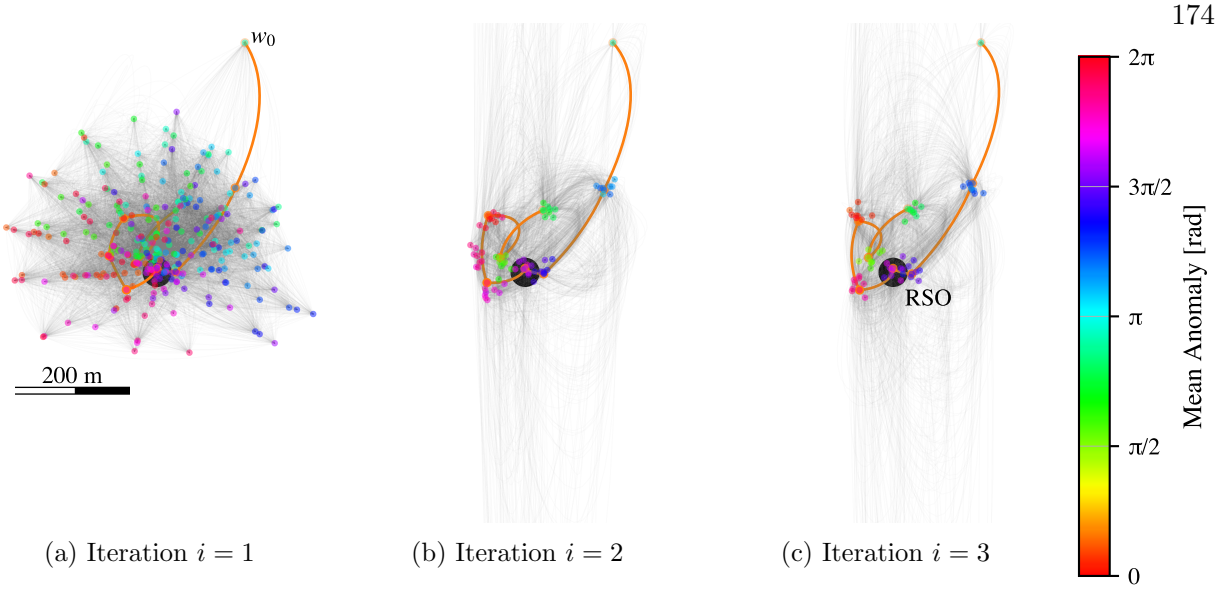
function PLANINSPECTIONTRAJECTORY($\boldsymbol{r}_0, M_0, \mathcal{L}$)
  $w_0 \leftarrow (\boldsymbol{r}_0, M_0)$
  for $i \in 1, 2, 3, \ldots$ **do**
    if $i = 1$ **then**
      $\mathcal{W} \leftarrow \{w_0\} \cup$ SEEDWAYPOINTS($\mathcal{L}, N_{\text{seed}}$)
    else
      $\mathcal{W} \leftarrow \{w_0\} \cup$ RESAMPLEWAYPOINTS($\mathcal{W}_{\text{sol}}[1:], N_{\text{resample}}, \sigma_r, \sigma_M$)
    end if
    $\mathcal{E} \leftarrow$ CONNECTGRAPH($\mathcal{W}, N_{\text{conn}}$)
    $\boldsymbol{x} \leftarrow$ construct and solve MIQCQP (Equation 10.5-10.14)
    $\mathcal{W}_{\text{sol}} \leftarrow$ PARSESOLUTION($\boldsymbol{x}, \mathcal{W}$)
  end for
  return $\mathcal{W}_{\text{sol}}$
end function

Figure 10.1 shows three iterations of the method on an example case. The differences between first and later iteration waypoint sampling methods are clearly visible, and the incumbent solution quality improves with each iteration.

10.2.2 Graph Sampling and Construction

Three algorithms are used for constructing a discrete graph in the waypoint space. SEEDWAYPOINTS and RESAMPLEWAYPOINTS generate sets of waypoints for the PRM, and CONNECTGRAPH draws dynamically feasible edges between the sampled waypoints to produce a graph of possible trajectories.

Algorithm 16, SEEDWAYPOINTS, generates waypoints for the initial iteration of PLANINSPECTIONTRAJECTORY. While PRM-based methods generally sample the waypoint space uniformly, a large portion of the domain does not result in inspected points and is thus irrelevant to the task; including points from this region results in an intractably large optimization problem. Instead, N_{seed} waypoints are sampled for each RSO point in $\mathcal{L}$. The waypoint position $\boldsymbol{r}_i$ is sampled uniformly between the conjunction radius R_k and the maximum inspection radius $R_{\max}$ in the inspection point normal direction $\hat{\boldsymbol{n}}_p$. The mean anomaly M_i is sampled uniformly over times where the point



(a) Iteration $i = 1$ (b) Iteration $i = 2$ (c) Iteration $i = 3$

Figure 10.1: PRM solution iterations.

is illuminated.

In subsequent iterations, Algorithm 17, ResampleWaypoints, is used to generate new waypoints for the PRM in order to improve beyond the previous iteration's discretization. For each waypoint in the solution trajectory $\mathcal{W}_{\text{sol}}$, N_{resample} new waypoints are generated in addition to the waypoint from the incumbent solution. These are sampled from a normal distribution around the solution point with a standard deviation of σ_r in position and σ_M in true anomaly. Because the points in $\mathcal{W}_{\text{sol}}$ are included, the new roadmap will be at worst as good as the previous iteration.

Algorithm 18, ConnectGraph, builds the PRM by drawing edges between nearby waypoints. The structure of the problem is exploited to use the difference in mean anomaly as the distance metric. The modulus operator is applied to the mean anomaly difference to transform it into the range $(0, 2\pi]$ to reflect that any transition happens forward in time in at most one orbit. The N_{conn} closest CollisionFree child waypoints are connected from each waypoint, where collisions are checked at discrete points between each pair using the state transition matrix (STM) to solve the two-point boundary-value problem (Equation 10.4).

Algorithm 16 Generate N_{seed} waypoints with visibility of each illuminated point

function SEEDWAYPOINTS($\mathcal{L}, N_{\text{seed}}$)
  $\mathcal{W} \leftarrow \{\}$
  for $p_p \in \mathcal{L}$ **do**
    for $i \in 1, ..., N_{\text{seed}}$ **do**
      $\boldsymbol{r}_i \leftarrow \boldsymbol{r}_p + \hat{\boldsymbol{n}}_p \cdot \text{Uniform}(R_k - \|\boldsymbol{r}_p\|, R_{\max})$
      $M_i \leftarrow \text{Uniform}(M \in [0, 2\pi] \mid L(p_p, M) = \text{True})$
      $\mathcal{W} \leftarrow \mathcal{W} \cup \{(\boldsymbol{r}_i, M_i)\}$
    end for
  end for
  return $\mathcal{W}$
end function

10.2.3 Mixed-Integer Formulation

With the waypoints $\mathcal{W}$ and edges $\mathcal{E}$ computed, a MIQCQP can be formulated to find the optimal trajectory that satisfies the inspection percentage requirement. New notation is introduced for the optimization variables and weights:

- $\zeta \in [0, 1]$ is the fuel-time cost weight, with $\zeta = 1$ causing the optimizer to only minimize fuel use and $\zeta = 0$ causing the optimizer to only minimize inspection time.
- $x_{i,j}$ is a binary optimization variable that indicates whether the edge $(i \to j)$ is traversed in the solution. One $x_{i,j}$ is created for every edge in $\mathcal{E}$. The vector of all $x_{i,j}$ is $\boldsymbol{x}$.
- $\Delta v_{0,i}^{\text{lin}}$ is the Δv cost of the initial maneuver to travel from w_0 to w_i.
- $\Delta v_{i,j,k}^{\text{quad}}$ is the Δv cost of maneuver that must take place at w_j when coming from w_i heading to w_k via w_j.
- z_p is a binary optimization variable indicating if inspection point p on the RSO is inspected in the solution. One z_p is created for every point in the illuminated set $\mathcal{L}$. The vector of all z_p is $\boldsymbol{z}$.
- $\mathbb{I}$ is the inspectability matrix, which is size $\text{len}(\boldsymbol{z}) \times \text{len}(\boldsymbol{x})$. It is structured such that the entry at the row corresponding to z_p and the column corresponding to $x_{i,j}$ indicates

Algorithm 17 Generate N_{resample} waypoints around each waypoint in the solution

```
function ResampleWaypoints(W_sol, N_resample, σ_r, σ_M)
    W ← {}
    for w_w ∈ W_sol do
        W ← W ∪ {w_w}
        for i ∈ 1, ..., N_resample do
            do
                r_i ← Normal(r_w, σ_r I_{3×3})
            while ‖r_i‖ < R_k
            M_i ← Normal(M_w, σ_M)
            W ← W ∪ {(r_i, M_i)}
        end for
    end for
    return W
end function
```

whether point p of the RSO is inspectable on the edge between w_i and w_j:

$$\mathbb{I}_{p,(i,j)} = \bigvee_{M=M_i}^{M_j} I(p, \boldsymbol{r}_{i\to j}(M), M) \tag{10.1}$$

where $\boldsymbol{r}_{i\to j}(M)$ is the position along the trajectory between two waypoints w_i and w_j as a function of mean anomaly M.

The linear and quadratic Δv costs are given by

$$\Delta v_{0,i}^{\text{lin}} = \|\boldsymbol{v}_0 - \boldsymbol{v}_{\text{in}}(w_0, w_i)\| \tag{10.2}$$

$$\Delta v_{ijk}^{\text{quad}} = \|\boldsymbol{v}_{\text{out}}(w_i, w_j) - \boldsymbol{v}_{\text{in}}(w_j, w_k)\| \tag{10.3}$$

To compute the incoming $\boldsymbol{v}_{\text{in}}$ and outgoing $\boldsymbol{v}_{\text{out}}$ velocities for the trajectory edge between w_i and w_j, the equation

$$\begin{bmatrix} \boldsymbol{r}_j \\ \\ \boldsymbol{v}_{\text{out}} \end{bmatrix} = \boldsymbol{\Phi}_c^x\big(M_i + \big((M_j - M_i) \bmod 2\pi\big), M_i\big) \begin{bmatrix} \boldsymbol{r}_i \\ \\ \boldsymbol{v}_{\text{in}} \end{bmatrix} \tag{10.4}$$

is solved for $\boldsymbol{v}_{\text{in}}$ and $\boldsymbol{v}_{\text{out}}$, given that $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ are known.

The complete MIQCQP for path planning over a set of waypoints $\mathcal{W}$ and edges $\mathcal{E}$ is formulated in Equation 10.5 through 10.14. It is assumed that all summations are over the indices for which

Algorithm 18 Add edges between nearby waypoints

```
function ConnectGraph(𝒲, N_conn)
    ℰ ← {}
    for w_i ∈ 𝒲 do
        for w_j ∈ 𝒲 do
            d_j ← (M_j − M_i) mod (0, 2π]
        end for
        for N_conn smallest d_j where CollisionFree(w_i, w_j) do
            ℰ ← ℰ ∪ {(w_i → w_j)}
        end for
    end for
    return ℰ
end function
```

the respective variables are explicitly defined (i.e. $\sum_i \sum_j x_{i,j}$ only sums over existing edges).

$$\text{minimize} \quad \zeta \left(\sum_i \Delta v_{0,i}^{\text{lin}} x_{0,i} + \sum_i \sum_j \sum_k x_{i,j} \Delta v_{i,j,k}^{\text{quad}} x_{j,k} \right) + \frac{1-\zeta}{n} \sum_i \sum_j \Delta M_{i,j} x_{i,j} \tag{10.5}$$

$$\begin{aligned}
\text{subject to} \quad & x_{i,j} \in \{0,1\} && \forall\ (w_i \to w_j) \in \mathcal{E} && (10.6)\\
& \sum_k x_{j,k} \leq \sum_i x_{i,j} + b \leq 1 && \forall\ j;\ b = 1 \text{ if } j = 0 \text{ else } b = 0 && (10.7)\\
& n_i \in \{0, ..., n_{\max}\} && \forall\ w_i \in \mathcal{W};\ n_{\max} = 10 && (10.8)\\
& 0 \leq \big(2\pi(n_j - n_i) + M_j - M_i\big) x_{i,j} \leq 2\pi && \forall\ (w_i \to w_j) \in \mathcal{E} && (10.9)\\
& x_{0,i} = 0 && \forall\ i \mid \Delta v_{0,i}^{\text{lin}} > \Delta v_{\max} && (10.10)\\
& x_{i,j} + x_{j,k} \leq 1 && \forall\ i,j,k \mid \Delta v_{i,j,k}^{\text{quad}} > \Delta v_{\max} && (10.11)\\
& z_p \in \{0,1\} && \forall\ p \in \mathcal{L} && (10.12)\\
& \mathbb{I}\boldsymbol{x} \geq -\mathbb{M}(1 - \boldsymbol{z}) && \mathbb{M} \gg 1 && (10.13)\\
& \sum_p z_p \geq 0.9|\mathcal{L}| && && (10.14)
\end{aligned}$$

The objective function (Equation 10.5) consists of two terms: the fuel cost term, weighted by ζ, and the time cost, weighted by $1-\zeta$ and converted to time by division by the mean motion n. The fuel cost consists of a linear term for the cost of the initial maneuver based on which outgoing edge

from w_0 is selected and a quadratic term based on the necessary incoming and outgoing velocities at each visited waypoint. The time cost sums the ΔM between waypoints on traveled edges.

Constraints 10.6 through 10.9 are used to guarantee a single trajectory originating at w_0. Equation 10.7 is the traditional traveling salesman problem (TSP) in-out constraint: edges can only be traveled if they come from the starting waypoint ($b = 1$) or another traveled edge ($b = 0$), and the path cannot branch or merge. However, that alone allows for disconnected cycles in the solution. To prevent these, Equation 10.9 is introduced, which assigns an integer orbit number n_i to each waypoint w_i and requires that for sequential waypoints in a solution trajectory, the absolute mean anomaly always increases with a difference in $(0, 2\pi]$. As a result, no cycles can be formed and the less-than-one-orbit connection is used.

Constraints 10.10 and 10.11 are added for every Δv that exceeds $\Delta v_{\max}$. For the linear objective term, the corresponding constraint (10.10) trivially sets infeasible edges to be unselected; for the quadratic objective, the corresponding constraint (10.11) allows for incoming or outgoing edges to be selected but prevents both from being selected for maneuvers that are infeasible.

Finally, constraints 10.12 through 10.14 use a big-$\mathbb{M}$ formulation to ensure that at least 90% of illuminated points are inspected by the trajectory. For a trajectory with traveled edges encoded by $\boldsymbol{x}$, $\mathbb{I}\boldsymbol{x}$ results in a vector that is 0 for uninspected points and ≥ 1 for inspected points. The big-$\mathbb{M}$ formulation, with $\mathbb{M}$ arbitrarily set to 10000, allows for a minimum threshold on the number of inspected points to be a constraint.

10.2.4 Parsing the Optimization Solution

Algorithm 19 gives the method for extracting the sequence of waypoints in the solution from the optimization variable $\boldsymbol{x}$. When executing the plan in the high-fidelity simulator, the 2-point boundary value problem in Equation 10.4 is solved for each maneuver according to the current true state to reach the next waypoint. This prevents perturbations from compounding over the course of execution.

Algorithm 19 Parse the MIQCQP solution

```
function ParseSolution(x, W)
    W_sol ← [w_0]
    while True do
        w_i ← W_sol[−1]
        for j | x_{i,j} ∈ x do
            if x_{i,j} = 1 then
                Append(W_sol, w_j)
                break
            end if
        end for
        else                                  ▷ Return if no w_j is connected from w_i
            return W_sol
    end while
end function
```

10.2.5 Algorithm Parameters

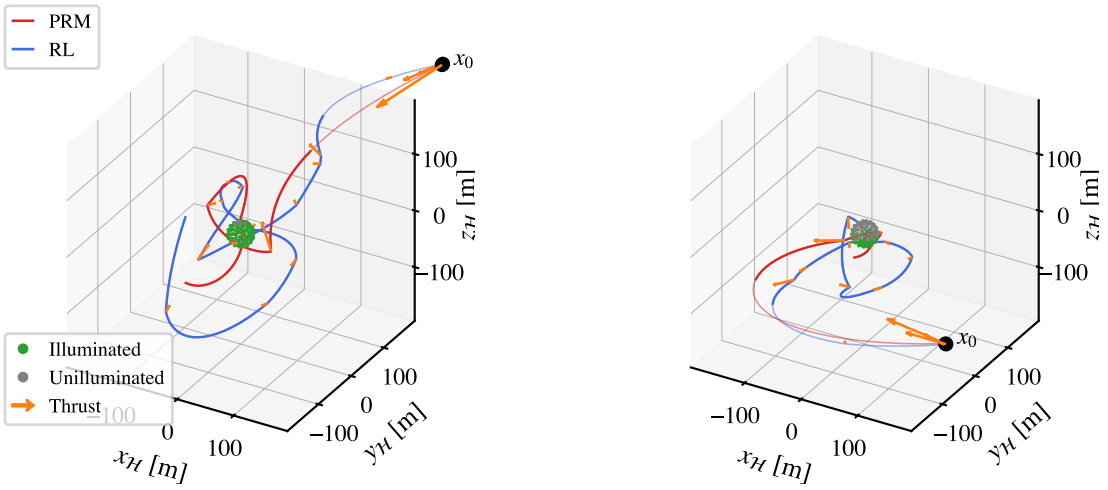
When evaluating the algorithm, the following parameters are used: $N_{\text{conn}} = 40$, $N_{\text{seed}} = 5$, $N_{\text{resample}} = 8$, $\sigma_r = 10$ m, $\sigma_M = 0.2$, and iterations $= 3$. For iterations $i \geq 2$, the solver can be warm-started with the previous iteration's solution. The problem is likely too large to solve exactly in reasonable time, so a non-optimal but feasible incumbent may be accepted instead. In the first iteration, the solver is run for 120 seconds beyond when the first incumbent solution is found; on the following warm-started iterations, the solver is run for 120 seconds per iteration or until the optimal solution is found. Gurobi is used to solve the MIQCQP [145].

10.3 Comparison of PRMs and RL for Inspection

The benchmarks of the five unshielded RL policies $\pi_{\text{single},\alpha}$ from chapter 9 for differing α are compared to a benchmark of the PRM solver over a range of object weights ζ. The Pareto fronts yielded by the two methods are compared and example trajectories are given.

10.3.1 Example Trajectories

Figure 10.2 compares the PRM solution with $\zeta = 0.5$ to the unshielded RL policy $\pi_{\text{single},\alpha=0.10}$ for the same cases tested in Figure 9.6 and Figure 9.7. The solutions are qualitatively similar. The



(a) $\beta = 7.6°$ case.

(b) $\beta = -69.2°$ case.

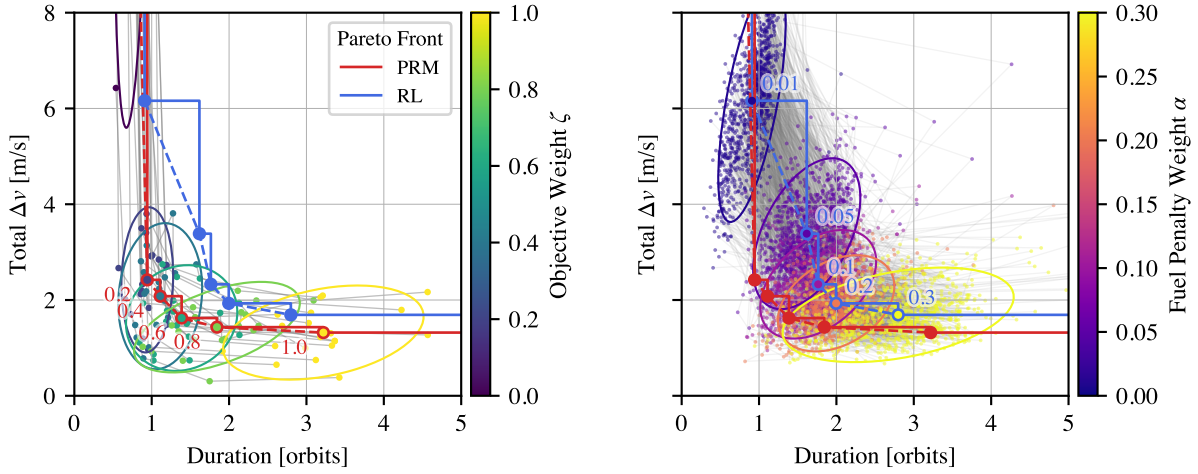
Figure 10.2: Comparison of PRM and RL trajectories on two sets of initial conditions in the $N = 1$ servicer environment.

most notable difference is that the PRM is more "comfortable" with trajectories that get very close to the RSO, while the RL solution is more conservative due to the collision penalty that it encounters during learning.

10.3.2 Pareto Fronts

Figure 10.3 shows the mean Pareto front for the PRM solver across ζ for $N = 20$ seeds and the RL solutions across α for $N = 1000$ seeds. Both methods produce reasonable results, clearly showing the time-fuel tradeoff as the respective weighting parameter is changed. The PRM solver shows stronger performance than the RL solution across the front.

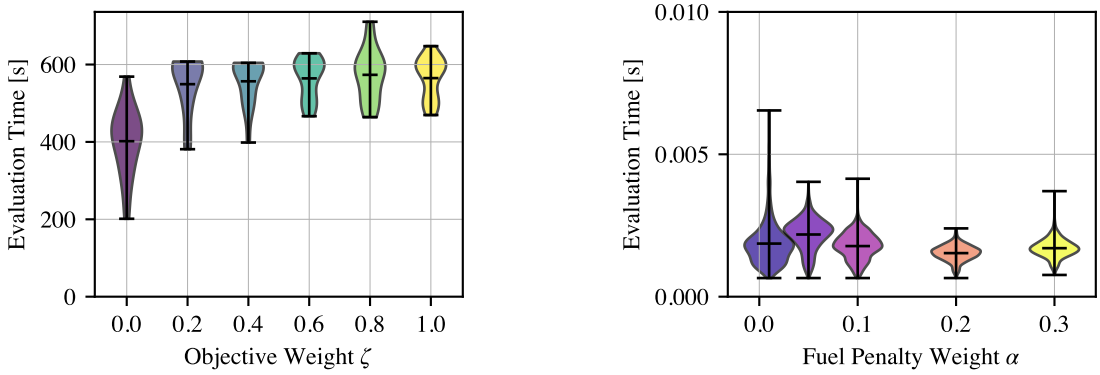
While the RL Pareto front is expectedly dominated by the PRM solver, this comes at significant computational—and thus operational—cost. Figure 10.4 compares the execution times of each method: The PRM method takes about 10 minutes on an Apple M2 Pro with 12 cores to converge, using the Gurobi solver that cannot be (trivially) used on a spacecraft computer. Computing these trajectories onboard is therefore infeasible with typical spacecraft resources, requiring a paradigm of offline planning and open-loop execution. Comparatively, the RL solution is very



(a) PRM with varying ζ, $N = 20$.

(b) RL with varying α, $N = 1000$.

Figure 10.3: Mean Pareto fronts across N trials; 2σ covariances.



(a) PRM, open loop.

(b) RL, closed loop.

Figure 10.4: Computation times; colors as in Figure 10.3.

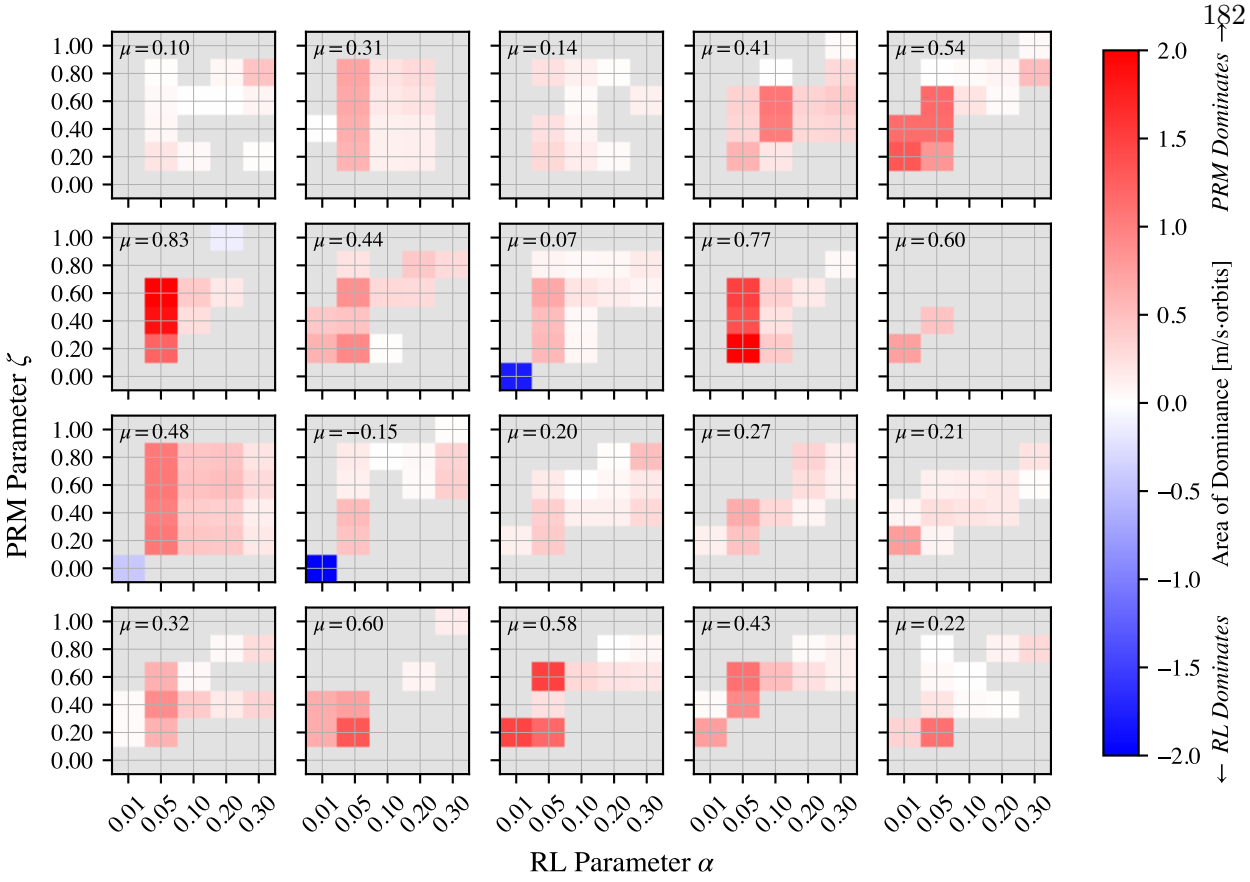


Figure 10.5: Area of domination between RL with varying α and PRM with varying ζ over 20 seeds; gray when neither method dominates.

cheap to evaluate due to its representation as a small (2×256 node) multilayer perceptron (MLP), which requires a trivial amount of storage and compute; even larger and more complex network architectures that would reasonably be considered for this problem have low enough inference costs to be usable onboard. As a result, it operates in a closed-loop, online manner. Not only does this allow for execution at arbitrary times, if some part of the RSO needed to be dynamically reinspected, the RL solution could trivially handle that case while the PRM would need an additional planning cycle.

10.3.3 Case-by-Case Performance Comparison

While the Pareto fronts quantify the mean performance difference between RL and PRMs, a case-by-case performance comparison is also informative. In Figure 10.5, the performance of RL is compared to the PRM solver for each combination of α and ζ for 20 seeds (one per subplot). The area of the rectangle between the two methods on the fuel-time axes in m/s·orbits is plotted (positive if PRM dominates, negative if RL) and is left gray if neither method dominates. The mean μ area of dominating rectangles for the seed is also given.

Domination tends to only occur when α and ζ induce solutions at similar points along the Pareto fronts. Across much of the fronts, the size of domination is relatively low, reflecting minimal losses in time or fuel when using RL instead of PRMs, aside from the other benefits of the RL-based approach. However, the average domination is often skewed higher due to greater domination around $\alpha = 0.05$, which reflects some challenges in obtaining low-time solutions with RL.

As a result, the tradeoff between the two methods is autonomy and responsiveness at the cost of some additional fuel or time. For a science or security mission with opportunistic events, those benefits of RL may outweigh the marginal costs. In scenarios such as deep space missions where resources are limited and ample planning time is available, the PRM method is superior.

Chapter 11

Conclusion

11.1 Summary of Contributions

This dissertation advances the state of the art in applying reinforcement learning (RL) to problems in spacecraft planning, scheduling, and control, yielding new autonomous capabilities. Consideration is given to all aspects of the methods, probing questions such as:

- How can realistic learning environments be formulated?
- How should training pipelines be set up?
- How do policies perform across deployment conditions? Under out-of-distribution conditions?
- How can safety constraints be enforced?
- When can apples-to-apples comparison be performed between RL and other methods?

These questions are answered for variations of the agile Earth-observing satellite scheduling problem (AEOSSP) and the resident space object (RSO) inspection problems. Methodologically, they also offer a framework for analyzing the application of RL to other spacecraft tasking problems.

11.1.1 Earth Observation Scheduling

A realistic environment for the AEOSSP is formulated in chapter 3. The problem is solved in two ways: a mixed-integer linear program (MILP) approach modified to produce optimal solu-

tions in the realistic environment (chapter 4) and a deep reinforcement learning (DRL) approach (chapter 5).

The MILP formulation advances the state of the art by efficiently accounting for true slew transition times (as opposed to using a lower-order heuristic) and by studying large instances of the problem with thousands of requests not considered elsewhere in the literature. Insights into the difficulty of different scenarios are gained from the benchmarks, providing trends that are generally applicable to the problem. As with other efficiently designed ground-based planners, this method is practical and performant for offline planning but too computationally expensive for onboard planning. The high quality of the plans produced because of accurate dynamics learned from a model of the system means that replanning on the fly necessitated by plan infeasibility is close to zero, even when generating very aggressive plans.

The RL-based approach to the AEOSSP is developed and compared to the MILP solver, which returns a best incumbent solution and a bound on optimality. After performing ablation studies to establish the necessity of a semi-Markov decision process (sMDP) formulation of the problem, the resulting policies are shown to be competitive with optimal solutions to the AEOSSP; these policies have the benefits of being closed-loop and executable onboard as opposed to computationally expensive, open-loop schedules. When compared in a high-fidelity simulation environment across a range of request densities and distributions, policies trained with RL performed between optimal and 22% below the optimal solution bound found using a computationally expensive MILP solver. In some cases, the policy found superior solutions with real-time closed-loop planning than the preplanner could find in an hour of computation. Also, an advantage of RL over other planning abstractions is demonstrated: implementing arbitrary resource constraints on the problem is relatively easy. While certain constraints can be challenging to fully implement in other optimization frameworks, RL can be trained with awareness of these constraints as long as they are present in the simulator. When combined with a shield to guarantee operational safety, the policy is shown to be able to leverage its understanding of safety considerations to minimize the impact of a power constraint on performance toward the objective by entering a charging state when fewer rewards

are available.

Chapter 6 develops how single-agent policies learned for the AEOSSP with DRL can generalize to the constellation scheduling problem through the use of information-sharing algorithms, resulting in a scalable, closed-loop, constellation-wide planning method. By comparing a variety of information-sharing algorithm variations, algorithms that share intent—i.e., preemptively alerting other satellites to the fact that a task is claimed—are most effective; they result in the highest reward and lowest duplication of effort, while also more evenly distributing the task load among members of the constellation. The autonomous methods are again compared to MILP solutions to quantify optimality and relative computational cost. Across a range of small-to-medium test cases, the policies perform at worst 90% as well as the 2-hour MILP solutions, and often outperform them. This trend is then extended to larger problems with more satellites and randomly appearing requests. Even when giving the MILP solver enough information for open-loop planning—as opposed to the policies' closed-loop processing of new requests as they appear—the MILP solver struggles: when solving for constellations of 16 or more satellites, no acceptable solution is found even with hours of compute time. Comparatively, the policies with information sharing do not suffer from scalability limitations.

Finally, the tip-and-cue problem formulation (chapter 7) demonstrates how new capabilities can be unlocked with distributed autonomy via RL. This approach is necessary, as traditional preplanning methods are unable to account for target locations that are dynamically revealed. Per-satellite, closed-loop scheduling policies are learned with centralized training and decentralized execution such that they scale to arbitrary constellations while maintaining strong performance. When studying the performance of these policies, there is significant evidence that agents have learned to actively collaborate with each other. As satellites are added to a constellation, the performance of the satellites working together grows faster than the number of satellites. Individually, the policy yields diverse behaviors among the agents: depending on the location of a satellite within the constellation, it may assume a specific role that biases it toward scanning or imaging.

11.1.2 Spacecraft Inspection

The treatment of the RSO inspection problem (chapter 8) with impulsive maneuvers follows a similar path. An RL-based approach for autonomous inspection is developed with analytical guarantees for safety (chapter 9). Then, a probabilistic roadmap (PRM)-based preplanning approach is designed (chapter 10) and the two are compared on the fuel-time Pareto front. Both methods are novel in that they leverage natural relative motion to yield low-fuel trajectories.

RL yields autonomous inspection trajectories, generalizing well to different orbits and initial conditions. Depending on the fuel-time tradeoff that the policy was trained for, inspection takes about 0.5 to 3 orbits and 1 to 8 m/s of fuel. While the policies generally produce safe and feasible trajectories, a shield is formulated for analytically passively safe impulsive maneuvers under Clohessy-Wiltshire-Hill (CWH) assumptions and is generalized to eccentric orbits and multiple agents. Tests of the shield show limited impacts on performance while yielding analytically expected safety. With the RSO geometry considered in this formulation, having multiple servicers does not yield a performance advantage due to competition for limited tasks; future work discusses opportunities where multi-agent inspection may be fruitful.

The preplanning approach that is developed for the inspection problem using PRMs and mixed-integer quadratically-constrained quadratic programs (MIQCQPs) is novel in its own right and provides a comparison for the performance of the RL-based policies. Comparing the methods on the fuel-time Pareto front, the PRM method tends to dominate, but only marginally. If on-demand autonomous inspection is necessary, such as in changing or low-communication environments, the RL-based policies may be worth slightly increased costs; however, in high-information environments with sufficient planning time, PRMs yield mild savings in time and fuel.

11.2 Future Work

There are numerous applications for RL in the context of space missions; this dissertation considered only a subset of two classes of problems. While some refinement of RL for these problems

is still possible, future research should also take these results as evidence that RL is a viable approach for the broader field of spacecraft planning, scheduling, and control and focus on enabling new capabilities for problem types that RL is particularly well-suited to handle: environments that are uncertain, adversarial, and changing necessitate methods with closed-loop autonomy.

While the basic AEOSSP is well studied for both traditional methods and now RL, numerous variations of it exist. These introduce practical challenges that operators face but are not often included in the literature and unlock new capabilities that rely on emerging, underutilized technologies. Uncertainty due to cloud coverage is one of the most impactful factors in Earth-observing satellite (EOS) operations. While static plans can at best incorporate a forecast in the planning pipeline, onboard image-processing capabilities allow for active replanning based on whether the region of interest was successfully captured. This paradigm is not new [11], but there is potential to significantly expand upon it: within a constellation, satellites should be able to share information about observed clouds in order to improve the chance of success of others' images. For some requests, small changes in time or viewing angle could lead to successful collection, while other requests may appear to have such a low chance of success that they are better left until later. Another poorly studied case that is significant to operators is requests with multi-image dependencies: these include stereo imaging and periodic monitoring, both of which lack many traditional or RL-based treatments.

More generally, the tip-and-cue formulation presented in chapter 7 is but a basic treatment of a previously unstudied but imminently possible paradigm for Earth observation. Applications range from finding and monitoring natural phenomena on Earth and other planets to commercial and defense uses. A cursory attempt at this for wildfire monitoring was presented in Reference 101. To advance this problem, the capability of generating a surrogate model that maps regions yielding high rewards should be developed. An interesting variation of the problem reframes it as a monitoring task, in which satellites must monitor interesting regions that intermittently change.

The RSO inspection task is generally a less mature problem than Earth observation. As such, there are many avenues of further research open. The basic problem presented in chapter 8 and

its continuous-control counterpart should be solved using other methods from the fields of path planning and optimal control to better understand the fuel-time tradeoff. More complex RSOs should be considered: while the RSO used in chapter 9 was too small to benefit from multiple servicers, a space-station- or asteroid-sized RSO may see benefits with multiple servicers. More complex RSOs also add the challenges of self-shadowing, self-occlusion, and time-varying geometry. While some research has considered a degree of shape reconstruction [84, 164], tighter incorporation of those algorithms into the planning pipeline would be beneficial. Another challenge is moving the problem to an adversarial setting: the RSO must try to complete a task while also keeping its payload hidden from the servicer. While current research tends to assume that the servicers start in the region of the RSO, the inspection problem could be incorporated into a larger rendezvous sequence planning problem: what RSOs should a servicer inspect, and when?

Beyond the classes of problems considered in this dissertation, RL is promising and understudied for other spacecraft planning, scheduling, and control problems. Many deep-space missions are greatly limited by closing the communication loop with Earth; RL could improve the yield of many operations of this type. Growing interest in adversarial tasks in space, such as those in Reference 52, presents another front where autonomous decision-making is necessary. Finally, the application of RL to spacecraft operations is just reaching the maturity necessary for real-world use. There is much to be learned from attempting to bridge the sim-to-real gap when using RL-based policies onboard a real satellite.

Glossary

AEOCSP agile Earth-observing constellation scheduling problem 15, 43, 98

AEOS agile Earth-observing satellite xiii, 40, 43, 44, 98

AEOSSP agile Earth-observing satellite scheduling problem ii, viii, xi, xiii, 2, 10, 11, 13, 15, 42, 43, 46–51, 53–56, 60, 77–79, 82, 91, 93, 98–100, 115–117, 139, 171, 184–186, 188

BSM Backsubstitution method 22, 24, 25, 32

CBF control barrier function 11, 140, 141

CWH Clohessy-Wiltshire-Hill 10, 14, 139, 146–148, 150, 187

DAG directed acyclic graph 10, 53, 60

DCOP distributed constraint optimization problem 115

Dec-POMDP decentralized partially-observable Markov decision process 7

Dec-POsMDP decentralized partially-observable semi-Markov decision process 100, 118, 144

DRL deep reinforcement learning 8, 11, 14, 15, 36, 84, 85, 115, 120, 185, 186

EO-1 Earth Observing-1 4, 115

EOS Earth-observing satellite 43, 115, 161, 188

FOR field of regard 50, 116–118

FOV field of view 4, 49, 116–118, 143

FSW flight software xi, 21–25, 27, 28, 30–32, 34–38, 47, 82, 141

GAE generalized advantage estimation 8, 9, 81, 86

GNC guidance, navigation, and control 34

ILS iterative local search 3, 10, 12, 54, 77

ISAM in-space assembly and manufacturing 4

KSP Kerbal Space Program 21, 22

LEO low Earth orbit 116, 133–135, 141

LOS line-of-sight 131, 132

MARL multi-agent reinforcement learning xv, 15, 16, 120, 121, 169

MCTS Monte Carlo tree search 171

MDP Markov decision process xiii, 5–8, 11, 13–15, 20, 26, 31, 77–81, 86, 99, 100, 115, 118, 120, 141, 144, 171, 172

MILP mixed-integer linear program xi, xiv, 3, 10, 12, 15, 54, 56, 60–62, 64, 65, 67, 69, 71, 74, 76–79, 87–89, 91, 98, 106, 107, 111–113, 171, 184–186

MIP mixed-integer program 54

MIQCQP mixed-integer quadratically-constrained quadratic program 171–173, 175, 176, 179, 187

MLP multilayer perceptron 11, 58, 121, 155, 182

MRP modified Rodrigues parameter 46

NN neural network 8, 55, 57, 58, 61, 62, 71, 140, 155

POMDP partially-observable Markov decision process 7, 26, 82

POsMDP partially-observable semi-Markov decision process 82, 93, 141

PPO proximal policy optimization 8, 9, 24, 81, 86, 120, 155, 165

PRM probabilistic roadmap xvii, 10, 16, 135, 159, 171–174, 179–183, 187

RL reinforcement learning ii, xi, xiv, xvii, 1, 5, 6, 8, 10–16, 20–24, 26, 36, 77–80, 82, 85–88, 91, 94, 97–99, 112, 113, 140, 141, 145, 155, 157, 171, 172, 179–189

RPO rendezvous and proximity operations 4, 133, 140, 171

RRT rapidly-exploring random tree 171, 192

RSO resident space object ii, xvi, 1, 4, 5, 13–16, 38–40, 133–144, 146–148, 150, 151, 159, 161, 170–173, 175, 176, 180, 182, 184, 187–189

SCP sequential convex programming 141

SLAM simultaneous localization and mapping 140

sMDP semi-Markov decision process 7, 15, 31, 78–81, 86, 144, 165, 185

SSO Sun-synchronous orbit 137

SSPICY Small Spacecraft Propulsion and Inspection Capability 2, 5

SST stable sparse rapidly-exploring random tree (RRT) 171, 172

STARS Safe Trusted Autonomy for Responsible Spacecraft 4

STM state transition matrix 150, 151, 174, 206, 208

TSP traveling salesman problem 10, 61, 178

Bibliography

[1] ESA's Annual Space Environment Report. Technical Report GEN-DB-LOG-00288-OPS-SD, ESA Space Debris Office, May 2026.

[2] Daniel Selva and David Krejci. A survey and assessment of the capabilities of Cubesats for Earth observation. Acta Astronautica, 74:50–68, May 2012.

[3] Gauthier Picard, Clément Caron, Jean-Loup Farges, Jonathan Guerra, Cédric Pralet, and Stéphanie Roussel. Autonomous Agents and Multiagent Systems Challenges in Earth Observation Satellite Constellations. In International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), pages 39–44, Richland, SC, May 2021. International Foundation for Autonomous Agents and Multiagent Systems.

[4] Planet Labs. Our Constellations. https://planet.com/insights/, April 2021.

[5] Getting SSPICY: NASA Funds Orbital Debris Inspection Mission - NASA, September 2024.

[6] Xinwei Wang, Guohua Wu, Lining Xing, and Witold Pedrycz. Agile Earth Observation Satellite Scheduling Over 20 Years: Formulations, Methods, and Future Directions. IEEE Systems Journal, 15(3):3881–3892, September 2021.

[7] Benedetta Ferrari, Jean-François Cordeau, Maxence Delorme, Manuel Iori, and Roberto Orosei. Satellite Scheduling Problems: A survey of applications in Earth and outer space observation. Computers & Operations Research, 173:106875, January 2025.

[8] Mike Safyan. Planet's Dove Satellite Constellation. In Joseph N. Pelton and Scott Madry, editors, Handbook of Small Satellites, pages 1057–1073. Springer International Publishing, Cham, 2020.

[9] Maxar Technologies. Our Constellation: The World's Most Advanced Earth Observation Satellites. https://www.maxar.com/maxar-intelligence/constellation.

[10] Space Development Agency. Tracking. https://www.sda.mil/tracking/.

[11] Gregg Rabideau, Daniel Tran, Steve Chien, Benjamin Cichy, and Rob Sherwood. Mission Operations of Earth Observing-1 with Onboard Autonomy.

[12] Ari Jonsson, Paul Morris, Nicola Muscettola, and Kanna Rajan. Next Generation Remote Agent Planner, 1999.

[13] S Chien, G Rabideau, R Knight, R Sherwood, B Engelhardt, D Mutz, T Estlin, B Smith, F Fisher, T Barrett, G Stebbins, and D Tran. ASPEN – Automated Planning and Scheduling for Space Mission Operations. 2000.

[14] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood. Casper: Space exploration through continuous planning. IEEE Intelligent Systems, 16(5):70–75, September 2001.

[15] Michael Seablom, Jacqueline Le Moigne, Sujay Kumar, Barton Forman, and Paul Grogan. Real-Time Applications of the Nasa Earth Science "New Observing Strategy". In IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium, pages 5317–5320, Kuala Lumpur, Malaysia, July 2022. IEEE.

[16] Alberto Candela, Jason Swope, and Steve A. Chien. Dynamic Targeting to Improve Earth Science Missions. Journal of Aerospace Information Systems, 20(11):679–689, November 2023.

[17] Steve Chien, Itai Zilberstein, Alberto Candela, David Rijlaarsdam, Tom Hendrix, Aubrey Dunne, Oriol Aragon, and Juan Puig Miquel. Flight of Dynamic Targeting on CogniSAT-6, May 2025.

[18] Eric Vitug. Distributed Spacecraft Autonomy (DSA). http://www.nasa.gov/directorates/spacetech/game_changing_development/projects/dsa, April 2020.

[19] David Barnhart, Roger Hunter, Alan Weston, Vincent Chioma, Mark Steiner, and William Larsen. XSS-10 micro-satellite demonstration. In AIAA Defense and Civil Space Programs Conference and Exhibit, Huntsville,AL,U.S.A., October 1998. American Institute of Aeronautics and Astronautics.

[20] Rebecca Reesman and Andrew Rogers. Getting in Your Space: Learning from Past Rendezvous and Proximity Operations. Aerospace Corporation Center for Space Policy and Strategy, May 2018.

[21] Dr Dale Arney, John Mulvaney, Christina Williams, Christopher Stockdale, and Nathanael Gelin. In-space Servicing, Assembly, and Manufacturing (ISAM) State of Play. 2023.

[22] eoPortal. MEV-1 & 2 (Mission Extension Vehicle-1 and -2). https://www.eoportal.org/satellite-missions/mev-1#references.

[23] Astroscale. ELSA-M. https://astroscale.com/elsa-m/.

[24] Air Force Research Laboratory. STARS. https://afresearchlab.com/technology/stars/.

[25] Richard S. Sutton and Andrew Barto. Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts London, England, 2 edition, 2018.

[26] Steven Bradtke and Michael Duff. Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In Advances in Neural Information Processing Systems, volume 7. MIT Press, 1994.

[27] Kunal Menda, Yi-Chun Chen, Justin Grana, James W. Bono, Brendan D. Tracey, Mykel J. Kochenderfer, and David Wolpert. Deep Reinforcement Learning for Event-Driven Multi-Agent Decision Processes. IEEE Transactions on Intelligent Transportation Systems, 20(4):1259–1268, April 2019.

[28] Duncan Eddy and Mykel Kochenderfer. Markov Decision Processes For Multi-Objective Satellite Task Planning. In 2020 IEEE Aerospace Conference, pages 1–12, Big Sky, MT, USA, March 2020. IEEE.

[29] Afshin Oroojlooy and Davood Hajinezhad. A Review of Cooperative Multi-Agent Deep Reinforcement Learning, April 2021.

[30] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. Nature, 588(7839):604–609, December 2020.

[31] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 575(7782):350–354, November 2019.

[32] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation, January 2019.

[33] Yuxiang Yang and Deepali Jain. Agile and Intelligent Locomotion via Deep Reinforcement Learning, May 2020.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

[36] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In International Conference on Learning Representations, San Juan, Puerto Rico, October 2018.

[37] Virginie Gabrel, Alain Moulet, Cécile Murat, and Vangelis Th Paschos. A new single model and derived algorithms for the satellite shot planning problem using graph theory concepts. Annals of Operations Research, 69:115–134, 1997.

[38] Michel Lemaitre, Gerard Verfaillie, Frank Jouhaud, Jean-Michel Lachiver, and Nicolas Bataille. Selecting and scheduling observations of agile satellites. Aerospace Science and Technology, 6(5):367–381, September 2002.

[39] Sean Augenstein. Optimal Scheduling of Earth-Imaging Satellites with Human Collaboration via Directed Acyclic Graphs. The Intersection of Robust Intelligence and Trust in Autonomous Systems: Papers from the AAAI Spring Symposium, pages 11–16, 2014.

[40] Sreeja Nag, Alan S. Li, and James H. Merrick. Scheduling algorithms for rapid imaging using agile Cubesat constellations. Advances in Space Research, 61(3):891–913, February 2018.

[41] Doo-Hyun Cho, Jun-Hong Kim, Han-Lim Choi, and Jaemyung Ahn. Optimization-Based Scheduling Method for Agile Earth-Observing Satellite Constellation. Journal of Aerospace Information Systems, 15(11):611–626, November 2018.

[42] Junhong Kim, Jaemyung Ahn, Han-Lim Choi, and Doo-Hyun Cho. Task Scheduling of Agile Satellites with Transition Time and Stereoscopic Imaging Constraints. Journal of Aerospace Information Systems, 17(6):285–293, June 2020.

[43] Benjamin Bernhard, Changrak Choi, Amir Rahmani, Soon-Jo Chung, and Fred Hadaegh. Coordinated Motion Planning for On-Orbit Satellite Inspection using a Swarm of Small-Spacecraft. In 2020 IEEE Aerospace Conference, pages 1–13, March 2020.

[44] Timothy Lauinger and Steve Ulrich. Path Planning for Optimal Coverage of Orbiting Space Structures Using Lissajous Curves. In AAS/AIAA Space Flight Mechanics Meeting, January 2025.

[45] David L. Applegate, Robert E. Bixby, Vašek Chvatál, and William J. Cook. The Traveling Salesman Problem: A Computational Study. Princeton University Press, 2006.

[46] Mark A. Stephenson and Hanspeter Schaub. Optimal Agile Satellite Target Scheduling with Learned Dynamics. Journal of Spacecraft and Rockets, pages 1–12, October 2024.

[47] Joshua Aurand, Christopher Pang, Sina Mokhtar, Henry Lei, Steven Cutlip, and Sean Phillips. Assessing Autonomous Inspection Regimes: Active Versus Passive Satellite Inspection. In AIAA SCITECH 2025 Forum, January 2025.

[48] Sara Spangelo, James Cutler, Kyle Gilson, and Amy Cohn. Optimization-based scheduling for the single-satellite, multi-ground station communication problem. Computers & Operations Research, 57:1–16, May 2015.

[49] Christopher G. Valicka, Deanna Garcia, Andrea Staid, Jean-Paul Watson, Gabriel Hackebeil, Sivakumar Rathinam, and Lewis Ntaimo. Mixed-integer programming models for optimal constellation scheduling given cloud cover uncertainty. European Journal of Operational Research, 275(2):431–445, June 2019.

[50] Xinwei Wang, Yi Gu, Guohua Wu, and John R. Woodward. Robust scheduling for multiple agile Earth observation satellites under cloud coverage uncertainty. Computers & Industrial Engineering, 156:107292, June 2021.

[51] Shreeyam Kacker. Spacecraft Autonomy through Computer Vision and Onboard Planning. PhD thesis, MIT, Cambridge, Massachusetts, 2025.

[52] Ross Allen, Yaron Rachlin, Jessica Ruprecht, Sean Loughran, Jacob Varey, and Herbert Viggh. SpaceGym: Discrete and Differential Games in Non-Cooperative Space Operations. In 2023 IEEE Aerospace Conference, pages 1–12. Zenodo, 2023.

[53] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control Barrier Functions: Theory and Applications. In 2019 18th European Control Conference (ECC), pages 3420–3431, Naples, Italy, June 2019. IEEE.

[54] David Van Wijk, Kyle Dunlap, Manoranjan Majji, and Kerianne Hobbs. Safe Spacecraft Inspection via Deep Reinforcement Learning and Discrete Control Barrier Functions. Journal of Aerospace Information Systems, 21(12):996–1013, December 2024.

[55] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe Reinforcement Learning via Shielding. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1), April 2018.

[56] Robert Reed, Hanspeter Schaub, and Morteza Lahijanian. Shielded Deep Reinforcement Learning for Complex Spacecraft Tasking. In American Control Conference. arXiv, July 2024.

[57] Robert Reed and Morteza Lahijanian. Learning-Based Shielding for Safe Autonomy under Unknown Dynamics. In 2025 American Control Conference (ACC), pages 4940–4946, Denver, CO, USA, July 2025. IEEE.

[58] Andrew Harris, Trace Valade, Thibaud Teil, and Hanspeter Schaub. Generation of Spacecraft Operations Procedures Using Deep Reinforcement Learning. Journal of Spacecraft and Rockets, 59(2):611–626, March 2022.

[59] Adrien Hadj-Salah, Rémi Verdier, Clément Caron, Mathieu Picard, and Mikaël Capelle. Schedule Earth Observation satellites with Deep Reinforcement Learning, November 2019.

[60] Xuexuan Zhao, Zhaokui Wang, and Gangtie Zheng. Two-Phase Neural Combinatorial Optimization with Reinforcement Learning for Agile Satellite Scheduling. Journal of Aerospace Information Systems, 17(7):346–357, July 2020.

[61] Jie Chun, Wenyuan Yang, Xiaolu Liu, Guohua Wu, Lei He, and Lining Xing. Deep Reinforcement Learning for the Agile Earth Observation Satellite Scheduling Problem. Mathematics, 11(19):4059, September 2023.

[62] Antoine Jacquet, Guillaume Infantes, Nicolas Meuleau, Emmanuel Benazera, Stéphanie Roussel, Vincent Baudoui, and Jonathan Guerra. Earth Observation Satellite Scheduling with Graph Neural Networks, August 2024.

[63] Adam Herrmann and Hanspeter Schaub. Reinforcement Learning for the Agile Earth-Observing Satellite Scheduling Problem. IEEE Transactions on Aerospace and Electronic Systems, 59(5):1–13, October 2023.

[64] Peiyan Li, Huiquan Wang, Yongxing Zhang, and Ruixue Pan. Mission planning for distributed multiple agile Earth observing satellites by attention-based deep reinforcement learning method. Advances in Space Research, page S0273117724005465, June 2024.

[65] Haijiao Wang, Zhen Yang, Wugen Zhou, and Dalin Li. Online scheduling of image satellites based on neural networks and deep reinforcement learning. Chinese Journal of Aeronautics, 32(4):1011–1019, April 2019.

[66] Adam Herrmann, Mark A. Stephenson, and Hanspeter Schaub. Single-Agent Reinforcement Learning for Scalable Earth-Observing Satellite Constellation Operations. Journal of Spacecraft and Rockets, pages 1–19, November 2023.

[67] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. Reinforcement Learning for Solving the Vehicle Routing Problem. In NeurIPS, 2018.

[68] Andrew Harris, Thibaud Teil, and Hanspeter Schaub. Spacecraft Decision-Making Autonomy Using Deep Reinforcement Learning. In AAS Spaceflight Mechanics Meeting, Maui, Hawaii, January 2019.

[69] Adam Herrmann and Hanspeter Schaub. A Comparison Of Deep Reinforcement Learning Algorithms For Earth-Observing Satellite Scheduling. In AAS/AIAA Spaceflight Mechanics Meeting, Austin, TX, January 2023.

[70] Islam Nazmy, Andrew Harris, Morteza Lahijanian, and Hanspeter Schaub. Shielded Deep Reinforcement Learning for Multi-Sensor Spacecraft Imaging. In 2022 American Control Conference (ACC), pages 1808–1813, Atlanta, GA, USA, June 2022. IEEE.

[71] Daniel Morgan, Soon-Jo Chung, and Fred Y. Hadaegh. Model Predictive Control of Swarms of Spacecraft Using Sequential Convex Programming. Journal of Guidance, Control, and Dynamics, 37(6):1725–1740, November 2014.

[72] Daniel Morgan, Giri P Subramanian, Soon-Jo Chung, and Fred Y Hadaegh. Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming. The International Journal of Robotics Research, 35(10):1261–1285, September 2016.

[73] G. Gaias and S. D'Amico. Impulsive Maneuvers for Formation Reconfiguration Using Relative Orbital Elements. Journal of Guidance, Control, and Dynamics, 38(6):1036–1049, June 2015.

[74] Adam W. Koenig and Simone D'Amico. Robust and Safe N-Spacecraft Swarming in Perturbed Near-Circular Orbits. Journal of Guidance, Control, and Dynamics, 41(8):1643–1662, August 2018.

[75] Hanspeter Schaub and John L. Junkins. Analytical Mechanics of Space Systems. AIAA Education Series. American Institute of Aeronautics and Astronautics, Inc, Reston, VA, fourth edition edition, 2018.

[76] M. Sabatini, R. Volpe, and G.B. Palmerini. Centralized visual based navigation and control of a swarm of satellites for on-orbit servicing. Acta Astronautica, 171:323–334, June 2020.

[77] Yashwanth Kumar Nakka, Wolfgang Hönig, Changrak Choi, Alexei Harvard, Amir Rahmani, and Soon-Jo Chung. Information-Based Guidance and Control Architecture for Multi-Spacecraft On-Orbit Inspection. Journal of Guidance, Control, and Dynamics, 45(7):1184–1201, July 2022.

[78] William Kuhl, Jun Wang, Duncan Eddy, and Mykel Kochenderfer. Markov Decision Processes for Satellite Maneuver Planning and Collision Avoidance, January 2025.

[79] Adam Herrmann and Hanspeter Schaub. Reinforcement Learning for Small Body Science Operations. In AAS Astrodynamics Specialist Conference, Charlotte, North Carolina, August 2022.

[80] Joshua Aurand, Steven Cutlip, Henry Lei, Kendra Lang, and Sean Phillips. Exposure-Based Multi-Agent Inspection of a Tumbling Target Using Deep Reinforcement Learning, May 2023.

[81] Henry H. Lei, Matt Shubert, Nathan Damron, Kendra Lang, and Sean Phillips. Deep Reinforcement Learning For Multi-Agent Autonomous Satellite Inspection. In Matt Sandnas and David B. Spencer, editors, Proceedings of the 44th Annual American Astronautical Society Guidance, Navigation, and Control Conference, 2022, pages 1391–1412, Cham, 2024. Springer International Publishing.

[82] Henry Lei, Zachary S Lippay, Anonto Zaman, Joshua Aurand, Amin Maghareh, and Sean Phillips. Stability Analysis of Deep Reinforcement Learning for Multi-Agent Inspection in a Terrestrial Testbed. In AIAA SCITECH 2025 Forum, Orlando, FL, January 2025.

[83] Andrea Brandonisio, Michèle Lavagna, and Davide Guzzetti. Reinforcement Learning for Uncooperative Space Objects Smart Imaging Path-Planning. The Journal of the Astronautical Sciences, 68(4):1145–1169, December 2021.

[84] Andrea Brandonisio, Lorenzo Capra, and Michèle Lavagna. Deep reinforcement learning spacecraft guidance with state uncertainty for autonomous shape reconstruction of uncooperative target. Advances in Space Research, page S0273117723005276, July 2023.

[85] Tommaso Guffanti, Daniele Gammelli, Simone D'Amico, and Marco Pavone. Transformers for Trajectory Optimization with Application to Spacecraft Rendezvous. In 2024 IEEE Aerospace Conference, pages 1–13, Big Sky, MT, USA, March 2024. IEEE.

[86] Yuji Takubo, Tommaso Guffanti, Daniele Gammelli, Marco Pavone, and Simone D'Amico. Towards Robust Spacecraft Trajectory Optimization via Transformers, October 2024.

[87] Kyle Dunlap, Nathaniel Hamilton, and Kerianne L Hobbs. Deep Reinforcement Learning for Scalable Multiagent Spacecraft Inspection. In AAS/AIAA Space Flight Mechanics Meeting, January 2025.

[88] Margherita Piccinin, Paolo Lunghi, and Michèle Lavagna. Deep Reinforcement Learning-based policy for autonomous imaging planning of small celestial bodies mapping. Aerospace Science and Technology, 120:107224, January 2022.

[89] Zheng Chen, Hutao Cui, and Yang Tian. Autonomous Maneuver Planning for Small-Body Reconnaissance via Reinforcement Learning. Journal of Guidance, Control, and Dynamics, pages 1–13, May 2024.

[90] Charles E. Oestreich, Richard Linares, and Ravi Gondhalekar. Autonomous Six-Degree-of-Freedom Spacecraft Docking with Rotating Targets via Reinforcement Learning. Journal of Aerospace Information Systems, 18(7):417–428, July 2021.

[91] Kanta Prasad Sharma, Indradeep Kumar, Pavitar Parkash Singh, K. Anbazhagan, Hussain Mobarak Albarakati, Mohammed Wasim Bhatt, Avlokulov Anvar Ziyadullayevich, Arti Rana, and Sivasankari S. A. Advancing spacecraft rendezvous and docking through safety

reinforcement learning and ubiquitous learning principles. Computers in Human Behavior, 153:108110, April 2024.

[92] Lorenzo Federici, Andrea Scorsoglio, Alessandro Zavoli, and Roberto Furfaro. Meta-reinforcement learning for adaptive spacecraft guidance during finite-thrust rendezvous missions. Acta Astronautica, 201:129–141, December 2022.

[93] Alessandro Zavoli and Lorenzo Federici. Reinforcement Learning for Robust Trajectory Design of Interplanetary Missions. Journal of Guidance, Control, and Dynamics, 44(8):1440–1453, August 2021.

[94] Nicholas B. LaFarge, Daniel Miller, Kathleen C. Howell, and Richard Linares. Autonomous closed-loop guidance using reinforcement learning in a low-thrust, multi-body dynamical environment. Acta Astronautica, 186:1–23, September 2021.

[95] Nicholas B. LaFarge, Kathleen C. Howell, and David C. Folta. Adaptive closed-loop maneuver planning for low-thrust spacecraft using reinforcement learning. Acta Astronautica, 211:142–154, October 2023.

[96] Mark Stephenson and Hanspeter Schaub. Autonomous Tip-and-Cue Earth-Observing Constellation Tasking with Reinforcement Learning. In IEEE Aerospace Conference, Big Sky, MT, March 2026.

[97] Mark Stephenson and Hanspeter Schaub. Comparing Probabilistic Roadmaps and Reinforcement Learning for Relative Motion Inspection of Space Objects. In AAS Rocky Mountain GN&C Conference, Breckenridge, Colorado, January 2026.

[98] Mark Stephenson and Hanspeter Schaub. Safe, Autonomous Multi-Agent Inspection of Space Objects Leveraging Relative Orbit Dynamics. In Advanced Maui Optical and Space Surveillance Technologies Conference, Maui, Hawaii, September 2025.

[99] Mark Stephenson, Daniel Huterer Prats, and Hanspeter Schaub. Autonomous Satellite Inspection in Low Earth Orbit with Optimization-Based Safety Guarantees. In International Workshop on Planning & Scheduling for Space, Toulouse, France, April 2025.

[100] Mark Stephenson, Lorenzzo Mantovani, Anaïs Cheval, and Hanspeter Schaub. Quantifying the Optimality of a Distributed RL-Based Autonomous Earth-Observing Constellation. In AAS GN&C Conference, Breckenridge, CO, February 2025.

[101] Mark A Stephenson and Hanspeter Schaub. Scalable Autonomous Decentralized Constellation Tasking on Asynchronous Semi-Markov Decision Processes. In International Workshop on Satellite Constellations & Formation Flying, Kaohsiung, Taiwan, December 2024.

[102] Mark A Stephenson and Hanspeter Schaub. BSK-RL: Modular, High-Fidelity Reinforcement Learning Environments for Spacecraft Tasking. In 75th International Astronautical Congress, Milan, Italy, October 2024. IAF.

[103] Mark Stephenson, Lorenzzo Mantovani, and Hanspeter Schaub. Intent Sharing for Emergent Collaboration in Autonomous Earth Observing Constellations. In AAS GN&C Conference, Breckenridge, CO, Februrary 2, 2024.

[104] Mark Stephenson and Hanspeter Schaub. Reinforcement Learning For Earth-Observing Satellite Autonomy With Event-Based Task Intervals. In AAS Rocky Mountain GN&C Conference, Breckenridge, CO, February 2024.

[105] Mark Stephenson, Lorenzzo Mantovani, Sean Phillips, and Hanspeter Schaub. Using Enhanced Simulation Environments to Improve Reinforcement Learning for Long-Duration Satellite Autonomy. In AIAA Science and Technology Forum and Exposition (SciTech), Orlando, Florida, January 2024.

[106] Mark Stephenson and Hanspeter Schaub. Optimal Target Sequencing In The Agile Earth-Observing Satellite Scheduling Problem Using Learned Dynamics. In AAS/AIAA Astrodynamics Specialist Conference, Big Sky, MT, August 2023.

[107] Adam Herrmann, Mark Stephenson, and Hanspeter Schaub. Reinforcement Learning For Multi-Satellite Agile Earth Observing Scheduling Under Various Communication Assumptions. In AAS Rocky Mountain GN&C Conference, Breckenridge, CO, February 2023.

[108] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning That Matters. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1), April 2018.

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

[110] J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, and Praveen Ravi. PettingZoo: Gym for Multi-Agent Reinforcement Learning, October 2021.

[111] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. Journal of Artificial Intelligence Research, 47:253–279, June 2013.

[112] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents, November 2017.

[113] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 5026–5033, Vilamoura-Algarve, Portugal, October 2012. IEEE.

[114] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering Diverse Domains through World Models, January 2023.

[115] Matteo El-Hariry, Antoine Richard, and Miguel Olivares-Mendez. RANS: Highly-Parallelised Simulator for Reinforcement Learning based Autonomous Navigating Spacecrafts, October 2023.

[116] Andrej Orsula, Matthieu Geist, Miguel Olivares-Mendez, and Carol Martinez. Space Robotics Bench: Robot Learning Beyond Earth, September 2025.

[117] Patrick W. Kenneally, Scott Piggott, and Hanspeter Schaub. Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework. Journal of Aerospace Information Systems, 17(9):496–507, September 2020.

[118] NVIDIA. Isaac sim.

[119] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), volume 3, pages 2149–2154, Sendai, Japan, 2004. IEEE.

[120] Michael Kaup, Cornelius Wolff, Hyerim Hwang, Julius Mayer, and Elia Bruni. A Review of Nine Physics Engines for Reinforcement Learning Research, 2024.

[121] Cody Allard, Manuel Diaz Ramos, and Hanspeter Schaub. Computational Performance of Complex Spacecraft Simulations Using Back-Substitution. Journal of Aerospace Information Systems, 16(10):427–436, October 2019.

[122] John Alcorn, Cody Allard, and Hanspeter Schaub. Fully Coupled Reaction Wheel Static and Dynamic Imbalance for Spacecraft Jitter Modeling. Journal of Guidance, Control, and Dynamics, 41(6):1380–1388, June 2018.

[123] Cody Allard, Hanspeter Schaub, and Scott Piggott. General Hinged Rigid-Body Dynamics Approximating First-Order Spacecraft Solar Panel Flexing. Journal of Spacecraft and Rockets, 55(5):1291–1299, September 2018.

[124] João Vaz Carneiro, Cody Allard, and Hanspeter Schaub. General Dynamics for Single- and Dual-Axis Rotating Rigid Spacecraft Components. Journal of Spacecraft and Rockets, 61(4):1099–1113, July 2024.

[125] Leah Kiner, Hanspeter Schaub, and Cody Allard. Spacecraft Backsubstitution Dynamics with General Multibody Prescribed Subcomponents. Journal of Aerospace Information Systems, 22(8):703–715, August 2025.

[126] B. Tapley, J. Ries, S. Bettadour, D. Chambers, M. Cheng, F. Condi, and S. Poole. The GGM03 Mean Earth Gravity Model from GRACE. In AGU Fall Meeting Abstracts, 2007.

[127] Charles H. Acton. Ancillary data services of NASA's Navigation and Ancillary Information Facility. Planetary and Space Science, 44(1):65–70, January 1996.

[128] J. M. Picone, A. E. Hedin, D. P. Drob, and A. C. Aikin. NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. Journal of Geophysical Research: Space Physics, 107(A12), December 2002.

[129] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. In Proceedings of the 35th International Conference on Machine Learning, volume 80, pages 3053–3062, June 2018.

[130] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research, 22:1–8, 2021.

[131] Hanspeter Schaub and Scott Piggott. Speed-constrained three-axes attitude control using kinematic steering. Acta Astronautica, 147:1–8, June 2018.

[132] Lorenzzo Quevedo Mantovani and Hanspeter Schaub. Improving Robustness of Autonomous Earth-Observing Spacecraft Using Training Environment Enhancements. Journal of Aerospace Information Systems, pages 1–12, January 2026.

[133] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study, June 2020.

[134] Anaïs Cheval and Hanspeter Schaub. Reinforcement Learning with Hybrid Action Representation for Autonomous Strip Imaging Task Scheduling in High-Resolution Super-Agile Satellites. In AIAA SciTech, Orlando, Florida, January 2026.

[135] J. G. Walker. Satellite Constellations. Journal of the British Interplanetary Society, 37:559, December 1984.

[136] Duncan Eddy and Mykel J. Kochenderfer. A Maximum Independent Set Method for Scheduling Earth-Observing Satellite Constellations. Journal of Spacecraft and Rockets, 58(5):1416–1429, September 2021.

[137] Guansheng Peng, Reginald Dewil, Cédric Verbeeck, Aldy Gunawan, Lining Xing, and Pieter Vansteenwegen. Agile earth observation satellite scheduling: An orienteering problem with time-dependent profits and travel times. Computers & Operations Research, 111:84–98, November 2019.

[138] Bistra Dilkina and Bill Havens. Agile Satellite Scheduling via Permutation Search with Constraint Propagation. Actenum Corporation, May 2005.

[139] Tengyue Mao, Zhengquan Xu, Rui Hou, and Min Peng. Efficient Satellite Scheduling Based on Improved Vector Evaluated Genetic Algorithm. Journal of Networks, 7, March 2012.

[140] C. Verbeeck, K. Sörensen, E.-H. Aghezzaf, and P. Vansteenwegen. A fast solution method for the time-dependent orienteering problem. European Journal of Operational Research, 236(2):419–432, July 2014.

[141] Xiaolu Liu, Gilbert Laporte, Yingwu Chen, and Renjie He. An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. Computers & Operations Research, 86:41–53, October 2017.

[142] Xiaoyu Chen, Gerhard Reinelt, Guangming Dai, and Andreas Spitz. A Mixed Integer Linear Programming Model for Multi-Satellite Scheduling. European Journal of Operational Research, 275(2):694–707, June 2019.

[143] Sreeja Nag, Mahta Moghaddam, Daniel Selva, Jeremy Frank, Vinay Ravindra, Richard Levinson, Amir Azemati, Alan Aguilar, Alan Li, and Ruzbeh Akbar. D-SHIELD: Distributed Spacecraft with Heuristic Intelligence to Enable Logistical Decisions. In IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium, pages 3841–3844, Waikoloa, HI, USA, September 2020. IEEE.

[144] Robert R. Britney and Robert L. Winkler. Bayesian point estimation and prediction. Annals of the Institute of Statistical Mathematics, 26(1):15–34, December 1974.

[145] Gurobi Optimizer Reference Manual. Gurobi Optimization LLC, 2023.

[146] Antoni Viros Martin, Kewei Cheng, Amy Fang, Zhaoliang Zheng, Hadas Kress-Gazit, Ankur Mehta, Daniel Selva, and Yizhou Sun. Decentralized Context-Based On-Board Planning for Earth Observation Missions. In AIAA Scitech 2021 Forum, VIRTUAL EVENT, January 2021. American Institute of Aeronautics and Astronautics.

[147] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence, 112(1-2):181–211, August 1999.

[148] Daniel Selva, Alessandro Golkar, Olga Korobova, Ignasi Cruz, Paul Collopy, and Olivier de Weck. Distributed Earth Satellite Systems: What Is Needed to Move Forward? Journal of Aerospace Information Systems, 14:1–26, August 2017.

[149] Ben Gorr, Alan Aguilar Jaramillo, Huilin Gao, Daniel Selva, Ankur Mehta, Yizhou Sun, Vinay Ravindra, Cédric H. David, and George H. Allen. Decentralized Satellite Constellation Replanning for Event Observation. Journal of Spacecraft and Rockets, pages 1–19, January 2025.

[150] Alan Aguilar Jaramillo, Ben Gorr, Huilin Gao, Daniel Selva, Ankur Mehta, Yizhou Sun, Vinay Ravindra, Cédric H David, and George H Allen. Decentralized Consensus-based Algorithms for Satellite Observation Reactive Planning with Complex Dependencies. In AIAA SciTech Forum, Orlando, FL, January 2025. AIAA.

[151] Joshua Holder, Natasha Jaques, and Mehran Mesbahi. Multi Agent Reinforcement Learning for Sequential Satellite Assignment Problems. In NeurIPS, 2024.

[152] Mykel J. Kochenderfer. Algorithms for Decision Making. Massachusetts Institute of Technology, 2022.

[153] James Munkres. Algorithms for the Assignment and Transportation Problems. Journal of the Society for Industrial and Applied Mathematics, 5(1):32–38, March 1957.

[154] Shiloh Dockstader and Planet Labs. Future Trends In New Space: Automated Tip & Cue. 2021.

[155] Chong Wang, Jun Li, Ning Jing, Jun Wang, and Hao Chen. A Distributed Cooperative Dynamic Task Planning Algorithm for Multiple Satellites Based on Multi-agent Hybrid Learning. Chinese Journal of Aeronautics, 24(4):493–505, August 2011.

[156] Jonathan Bonnet, Marie-Pierre Gleizes, Elsy Kaddoum, Serge Rainjonneau, and Gregory Flandin. Multi-satellite Mission Planning Using a Self-Adaptive Multi-agent System. In 2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems, pages 11–20, September 2015.

[157] Shreya Parjan and Steve A. Chien. Decentralized Observation Allocation for a Large-Scale Constellation. Journal of Aerospace Information Systems, 20(8):447–461, August 2023.

[158] Itai Zilberstein, Ananya Rao, Matthew Salis, and Steve Chien. Decentralized, Decomposition-Based Observation Scheduling for a Large-Scale Satellite Constellation. Proceedings of the International Conference on Automated Planning and Scheduling, 34:716–724, May 2024.

[159] Steve Chien, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Rebecca Castano, Ashley Davis, Dan Mandl, Stuart Frye, Bruce Trout, Seth Shulman, and Darrell Boyer. Using Autonomy Flight Software to Improve Science Return on Earth Observing One. Journal of Aerospace Computing, Information, and Communication, 2(4):196–216, April 2005.

[160] Steve Chien, David Mclaren, Joshua Doubleday, Daniel Tran, Veerachai Tanpipat, and Royol Chitradon. Using Taskable Remote Sensing in a Sensor Web for Thailand Flood Monitoring. Journal of Aerospace Information Systems, 16(3):107–119, March 2019.

[161] Akseli Kangaslahti, Alberto Candela, Jason Swope, Qing Yue, and Steve Chien. Dynamic Targeting of Satellite Observations Incorporating Slewing Costs and Complex Observation Utility *. In 2024 IEEE International Conference on Robotics and Automation (ICRA), pages 4876–4882, Yokohama, Japan, May 2024. IEEE.

[162] Mark Andrew Stephenson, Lorenzzo Quevedo Mantovani, and Hanspeter Schaub. Learning Policies for Autonomous Earth-Observing Satellite Scheduling over Semi-MDPs. Journal of Aerospace Information Systems, 22(9):789–799, 2025.

[163] David van Wijk, Kyle Dunlap, Manoranjan Majji, and Kerianne L. Hobbs. Deep Reinforcement Learning for Autonomous Spacecraft Inspection using Illumination, August 2023.

[164] Patrick Quinn, Bala Prenith Reddy Gopu, George M Nehma, and Dr Madhur Tiwari. Simulation Based Reward Function Validation for Multi-Agent On Orbit Inspection. In AIAA SciTech Forum, Orlando, FL, January 2026.

[165] Hanna Krasowski, Jakob Thumm, Marlon Müller, Lukas Schäfer, Xiao Wang, and Matthias Althoff. Provably Safe Reinforcement Learning: Conceptual Analysis, Survey, and Benchmarking, November 2023.

[166] Ritwik Majumdar, David C. Sternberg, Keenan Albee, and Oliver Jia-Richards. Demonstration of the Dyna Reinforcement Learning Framework for Reactive Close Proximity Operations. In AIAA SCITECH 2025 Forum, Orlando, FL, January 2025. American Institute of Aeronautics and Astronautics.

[167] Yanbo Li, Zakary Littlefield, and Kostas E. Bekris. Asymptotically Optimal Sampling-based Kinodynamic Planning. The International Journal of Robotics Research, 35(5), February 2016.

[168] Markus Iversflaten, Alex Hansson, David Sternberg, Oliver Jia-Richards, and Keenan Albee. Robust Replanning for Multi-Agent SmallSat Inspection in Failure Scenarios. In AIAA SCITECH 2025 Forum, Orlando, FL, January 2025. American Institute of Aeronautics and Astronautics.

[169] Keenan Albee, David C Sternberg, Alexander Hansson, David Schwartz, Ritwik Majumdar, and Oliver Jia-Richards. Architecting Autonomy for Safe Microgravity Free-Flyer Inspection. In IEEE Aerospace Conference, March 2025.

Appendix A

Relative Motion Dynamics with an Eccentric Chief

The dynamics of relative motion in an eccentric orbit is derived in chapter 14.5 of Reference 75. For reference, the state transition matrix (STM) for that system is given here.

The STM is first found for the relative orbital elements coordinate set

$$\delta\boldsymbol{\alpha e} = \begin{bmatrix} \delta a & \delta\theta & \delta i & \delta q_1 & \delta q_2 & \delta\Omega \end{bmatrix}^T \tag{A.1}$$

and is of the form

$$\boldsymbol{\Phi}_c^{\delta\boldsymbol{\alpha e}}(t,t_0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ A & B & 0 & C_1 & C_2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

where

$$A = -\frac{3}{2}\frac{a\eta}{r^2}(M - M_0) \tag{A.3}$$

$$B = \left(\frac{r_0}{r}\right)^2 \tag{A.4}$$

$$C_1 = \frac{1}{r^2\eta^2}\Big[r\sin\theta\big(r + a(1-q_1^2)\big) - r_0\sin\theta_0\big(r_0 + a(1-q_1^2)\big) + aq_1q_2(r\cos\theta - r_0\cos\theta_0) + q_2(r-r_0)(a+r+r_0)\Big] \tag{A.5}$$

$$C_2 = \frac{1}{r^2\eta^2}\Big[-r\cos\theta\big(r + a(1-q_2^2)\big) + r_0\cos\theta_0\big(r_0 + a(1-q_2^2)\big) - aq_1q_2(r\sin\theta - r_0\sin\theta_0) - q_1(r-r_0)(a+r+r_0)\Big] \tag{A.6}$$

using the state of the chief c to compute A, B, C_1, and C_2.

To convert from relative orbital elements to Hill-frame Cartesian relative states

$$\boldsymbol{x}_{d/c} = \begin{bmatrix} \mathcal{H}\boldsymbol{r}_{d/c}^T \\ \\ \mathcal{H}\boldsymbol{v}_{d/c}^T \end{bmatrix} \tag{A.7}$$

the linearized mapping matrix $\boldsymbol{A}_c$ and inverse mapping matrix $\boldsymbol{A}_c^{-1}$ are used. These are

$$\boldsymbol{A}_c = \begin{bmatrix} \frac{r}{a} & \frac{rV_r}{V_t} & 0 & -\frac{r}{p}\left(2aq_1 + r\cos\theta\right) & -\frac{r}{p}\left(2aq_2 + r\sin\theta\right) & 0 \\ 0 & r & 0 & 0 & 0 & r\cos i \\ 0 & 0 & r\sin\theta & 0 & 0 & -r\cos\theta\sin i \\ -\frac{V_r}{2a} & \left(\frac{1}{r} - \frac{1}{p}\right)h & 0 & \frac{V_r aq_1 + h\sin\theta}{p} & \frac{V_r aq_2 - h\cos\theta}{p} & 0 \\ -\frac{3V_t}{2a} & -V_r & 0 & \frac{3V_t aq_1 + 2h\cos\theta}{p} & \frac{3V_t aq_2 + 2h\sin\theta}{p} & V_r\cos i \\ 0 & 0 & V_t\cos\theta + V_r\sin\theta & 0 & 0 & (V_t\sin\theta - V_r\cos\theta)\sin i \end{bmatrix} \tag{A.8}$$

and

$$\boldsymbol{A}_c^{-1} =$$

$$\begin{bmatrix}
2\alpha(2+3\kappa_1+2\kappa_2) & -2\alpha\nu(1+2\kappa_1+\kappa_2) & 0 & \frac{2\alpha^2\nu p}{V_t} & \frac{2a}{V_t}(1+2\kappa_1+\kappa_2) & 0 \\
0 & \frac{1}{r} & \frac{\cos\theta+\nu\sin\theta}{\tan i\, r} & 0 & 0 & -\frac{\sin\theta}{\tan i\, V_t} \\
0 & 0 & \frac{\sin\theta-\nu\cos\theta}{r} & 0 & 0 & \frac{\cos\theta}{V_t} \\
\frac{3\cos\theta+2\nu\sin\theta}{\rho r} & -\frac{\nu^2\sin\theta/\rho+q_1\sin(2\theta)-q_2\cos(2\theta)}{r} & -\frac{q_2(\cos\theta+\nu\sin\theta)}{\tan i\, r} & \frac{\sin\theta}{\rho V_t} & \frac{2\cos\theta+\nu\sin\theta}{\rho V_t} & \frac{q_2\sin\theta}{\tan i\, V_t} \\
\frac{3\sin\theta-2\nu\cos\theta}{\rho r} & \frac{\nu^2\cos\theta/\rho+q_2\sin(2\theta)+q_1\cos(2\theta)}{r} & \frac{q_1(\cos\theta+\nu\sin\theta)}{\tan i\, r} & -\frac{\cos\theta}{\rho V_t} & \frac{2\sin\theta-\nu\cos\theta}{\rho V_t} & -\frac{q_1\sin\theta}{\tan i\, V_t} \\
0 & 0 & -\frac{\cos\theta+\nu\sin\theta}{r\sin i} & 0 & 0 & \frac{\sin\theta}{V_t\sin i}
\end{bmatrix} \tag{A.9}$$

where V_r and V_t are the radial and tangential velocity of the chief, and

$$\alpha = \frac{a}{r} \tag{A.10}$$

$$\nu = \frac{V_r}{V_t} \tag{A.11}$$

$$\rho = \frac{r}{p} \tag{A.12}$$

$$\kappa_1 = \alpha\left(\frac{1}{\rho} - 1\right) \tag{A.13}$$

$$\kappa_2 = \frac{\alpha\nu^2}{\rho} \tag{A.14}$$

Thus, the STM for Cartesian relative states with a chief in an eccentric orbit is

$$\boldsymbol{\Phi}_c^x(t,t_0) = \boldsymbol{A}_c(t)\boldsymbol{\Phi}_c^{\delta\boldsymbol{\alpha e}}(t,t_0)\boldsymbol{A}_c^{-1}(t_0) \tag{A.15}$$

Appendix B

Funding and Support

This work was supported by NASA Space Technology Graduate Research Opportunity (NSTGRO) grant 80NSSC23K1182. Parts of this work were also supported by the Air Force Research Lab grant FA9453-22-2-0050.

Out of obligation and true thanks: This work utilized 609,338 CPU-hours (69.56 CPU-years) of compute on the Alpine high-performance computing resource at the University of Colorado Boulder. Alpine is jointly funded by the University of Colorado Boulder, the University of Colorado Anschutz, and Colorado State University and with support from NSF grants OAC-2201538 and OAC-2322260.