Spacecraft Autonomy through Computer Vision and Onboard Planning

by

Shreeyam Kacker

MEng Aeronautical Engineering, Imperial College London, 2020
S.M. Aeronautics and Astronautics, Massachusetts Institute of Technology, 2022

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

DOCTOR OF PHILOSOPHY IN AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2025

© 2025 Shreeyam Kacker. This work is licensed under a CC BY-NC-ND 4.0 license.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Shreeyam Kacker
Department of Aeronautics and Astronautics
August 18, 2025

Certified by: Kerri L. Cahoy
Professor of Aeronautics and Astronautics, Thesis Committee Chair

Certified by: Olivier de Weck
Professor of Astronautics and Engineering Systems, Thesis Committee Member

Certified by: Kiruthika Devaraj
VP of Avionics and Spacecraft Technology, Planet Labs, Thesis Committee Member

Certified by: Steve A. Chien
Technical Fellow, Artificial Intelligence, NASA JPL, Thesis Reader

Certified by: Işıl Demir
VP of Mission Systems, Planet Labs, Thesis Reader

Accepted by: Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Spacecraft Autonomy through Computer Vision and Onboard Planning

by

Shreeyam Kacker

Submitted to the Department of Aeronautics and Astronautics
on August 18, 2025 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN AERONAUTICS AND ASTRONAUTICS

ABSTRACT

Earth observation (EO) from satellite platforms has experienced widespread growth since the commercialization and widespread availability of data, and has had large impacts on applications such as agriculture, disaster monitoring, and defense and intelligence. Observing unpredictable phenomena is still challenging for EO missions due to long lead times from scheduling, uplinking, and executing the image capture onboard the spacecraft. This delay between planning and execution means that conditions can change in between them, causing a task to become unobservable and missed in the meantime, for example due to cloud cover obscuring a target. Dynamic tasking (DT) is a mission concept that aims to mitigate this unpredictability by moving autonomy onboard the spacecraft and quickly reacting to conditions as observed, using several potential perception sources. In this work, we consider DT as applied to a tasked Earth-observing satellite, whose goal is to image Earth's landmass at predefined targets. The considered goal in this work for DT and onboard autonomy is avoidance of cloud cover, which can cause up to 66% of imaging tasks to be occluded, but factoring real-world constraints on operationalization and onboard edge computing. Instead of using end-to-end learned methods, we build upon existing work on spacecraft scheduling, incorporating a mixed-integer linear program (MILP) scheduler as the primary scheduling algorithm. Rather than directly incorporating DT into a global problem, we instead develop a set of heuristics which can estimate the utility of lookahead actions. We construct these heuristics from two directions: one from a simplified and constrained version of the scheduling problem with order statistics, and a second using a convolutional neural network with large amounts of synthetically generated data. We also consider DT using information from meteorological satellites in geostationary Earth orbit (GEO), parameterizing information delay rather than performing detailed analysis of data pipelines. In cases where all tasks are equally valued, all DT cases tested, including both meteorological and vision cases, outperform the conventional scheduler across all trials, ranging between 40% and 100% increase in total schedule utility based on cloud-free captures, depending on the DT method used. In cases where tasks have Pareto-distributed utility, the gap between the omniscient and conventional schedule shrinks drastically, to within 4% of total utility, and only a single DT method outperforms the conventional schedule consistently, as the environment becomes significantly more challenging due to asymmetric upside and downside risk. We also present methods by which to fractionate global state such that data can be efficiently stored and

updated across a satellite constellation, allowing these heuristics to continue working across the constellation with minimal modification.

Thesis committee chair: Kerri L. Cahoy
Title: Professor of Aeronautics and Astronautics

Thesis committee member: Olivier de Weck
Title: Professor of Astronautics and Engineering Systems

Thesis committee member: Kiruthika Devaraj
Title: VP of Avionics and Spacecraft Technology, Planet Labs

Thesis reader: Steve A. Chien
Title: Technical Fellow, Artificial Intelligence, NASA JPL

Thesis reader: Işıl Demir
Title: VP of Mission Systems, Planet Labs

Contents

List of Figures

List of Tables

Funding

This research was sponsored by Planet Labs PBC and supervised by Dr. Kiruthika Devaraj. We are grateful to Planet Labs for their collaboration.

Acknowledgments

This thesis was made possible by many different people along the way.

Firstly, I owe a tremendous debt of gratitude to my advisor, Prof. Kerri Cahoy. Kerri first took me on as an undergrad many years ago. I am grateful for her taking a chance on me as a graduate student, and the unparalleled amount of care and support she has given over the years.

Thank you to Dr. Kiruthika Devaraj, she similarly took a chance on me both hiring me at Planet for the first time in 2022 and then choosing and organizing the funding of this work.

I would also like to thank the rest of my PhD committee: Prof. Olivier de Weck, Dr. Steve Chien, and Işil Demir for their time and support throughout this process.

Thank you to friends who have helped me through graduate school, including Andrew Reilley, Josef Biberstein, Luka Govedič. Thank you also to some amazing labmates, Adam Boldi, Mary Dahl, and James Dingley.

I would not be here without the love and support of my family, who have valued education above all, and extended great help studying in another country far from home.

Finally, and most importantly, I am eternally grateful to my wife, Marie Barton. She has been tirelessly helping me through the most difficult parts of my degree with no end to her kindness to me. Additionally, she has more than tolerated me while I have spent days writing this thesis.

Nomenclature

Abbreviations

request A latitude/longitude point to image.

access Overflight of a particular request, including time and roll angle info

LEO Low Earth orbit.

GEO Geostationary Earth orbit.

ISL Inter-satellite link.

FoR Field of regard.

FoV Field of view.

ECI Earth-centered inertial frame.

ECEF Earth-centered Earth-fixed frame.

LLA Latitude/longitude/altitude frame.

DAG Directed acyclic graph.

ASOP Adaptive stochastic orienteering problem.

MDP Markov decision process.

SMDP Semi-Markov decision process.

POMDP Partially observable Markov decision process.

Symbols

r Request tuple.

a Access tuple.

c Intrinsic access reward.

$c_{\mathbf{min}}$ Minimum intrinsic access reward.

κ Pareto distribution parameter.

u State-dependent access reward.

t Time.

$t_{\mathbf{horizon}}$ Time for point on the horizon to come to nadir.

$\boldsymbol{\sigma}$ Schedule.

$|\boldsymbol{\sigma}|$ Schedule length.

$J(\boldsymbol{\sigma})$ Schedule utility.

μ Gravitational parameter.

a Semi-major axis.

R_E Earth radius.

h Altitude.

v Velocity.

$v_{\mathbf{orbit}}$ Orbital velocity.

$v_{\mathbf{track}}$ Ground track velocity.

T Orbital period.

θ Off-nadir roll pointing angle.

θ_{FoR} Maximum off-nadir roll pointing angle.

β Agility parameter.

$\mathcal{K}$ Set of constraints.

$\mathcal{R}$ Set of imaging requests.

$\mathcal{A}$ Set of imaging accesses.

$\boldsymbol{\Theta}$ Euler angles attitude vector.

$\mathbf{r}$ points in ECI.

γ Earth mean rotation rate.

f_x, f_y Camera focal length in x, y.

c_x, c_y Camera co-ordinate offset in x, y.

$\mathbf{R}$ Camera rotation matrix.

$\mathbf{T}$ Camera translation vector.

$\mathbf{K}$ Camera intrinsics matrix.

$\mathbf{P}$ Camera extrinsics matrix.

$J_t^+(\boldsymbol{\theta})$ Schedule advantage due to lookahead action at time t.

$J_t^-(\boldsymbol{\theta})$ Schedule cost due to lookahead action at time t.

$\widehat{J_t^+}(\boldsymbol{\theta})$ Estimate of schedule advantage.

$\widehat{J_t^-}(\boldsymbol{\theta})$ Estimate of schedule opportunity cost.

ΔJ Net schedule utility due to lookahead action.

$\widehat{\Delta J}$ Estimate of net schedule utility.

g Minimum gap spacing for analytic heuristic.

L Interval length for analytic heuristic.

N Number of points for analytic heuristic.

M Chain length for analytic heuristic.

Chapter 1

Introduction

Earth Observation (EO) with spacecraft platforms refers to using any sensor, such as optical, radar, or other instrument to take measurements of the Earth. While initially developed for applications in meteorology and intelligence, there is now widespread adoption of EO to many industries, including agriculture, climate monitoring, disaster relief, insurance, finance, logistics, mining, and transportation.

Due to this large array of applications and the high costs of designing, developing, and launching EO satellites with high quality instruments, there is significantly more demand for imagery than there is capacity, causing imagery supply to be oversubscribed. Whether done by human operators or by algorithms, planning and resource allocation is required. Mission controllers or planning algorithms, or a mixture of the two allocate imaging based on a variety of factors, such as revenue, weather forecasting, and revisit capability.

In order to increase capacity constraints, more operators are adopting the use of spacecraft constellations, allowing for significantly more frequent revisits. Planet Labs is a notable example of EO imaging using constellations, their Dove and SuperDove EO constellation, initially launched in 2014, can take an image of the entire Earth's land mass at least once per day [1]. The distributed, constellation-based approach for Earth observation has been so successful that many other commercial providers have followed up as constellation-first approaches, such as BlackSky and Satellogic aiming for similarly large constellations, both over 50 satellites [2, 3].

Most modern EO satellites are still constrained to simply following scheduled instructions from an operator via ground station. Scheduling timelines along with delays in uplinking a schedule can result in executing a plan that is out of date by the time it is executed. This scheduling delay in the past was insignificant compared with the long revisit rates, however it is now increasingly the bottleneck for information latency. This low latency is additionally becoming a large value-add for EO data. Unpredictable phenomena such as cloud cover or fast-moving events such as wildfires and object tracking can render even the best schedules ineffective, which can result in a large misallocation of resources. Clouds alone cover approximately 66% of Earth's surface at any one time [4] and are difficult to predict, which result in large fractions of an EO spacecraft's surface imaging effort being wasted.

Autonomy in Earth Observation can increase the applicability of EO to areas where latency is significant. The computing and sensing technology behind the mobile phone revolution of the early 2000s combined with advancements in vision systems have resulted in spacecraft equipped with advanced onboard computing systems with thousands of times more processing power than early autonomous mission experiments. There is a large gap in how best to operationalize onboard compute sensing, which this thesis aims to address, using algorithms to enhance and balance tradeoffs between perception and execution.

1.1 Brief History of Earth-Observing Satellites

To understand the limitations of current Earth-Observing satellites and where onboard autonomy can make improvements, it's useful to look at the history of EO and planning algorithms.

1.1.1 Origins

The history of Earth Observation (EO) is inseparable from space exploration itself. While initial EO occurred on rocket-based platforms, shortly after the launch of the first satellite, Sputnik 1 in 1957, Explorer 6 became the first satellite to take an image from orbit in 1959 [5]. The applications of EO from satellite platforms were immediately apparent in meteorology—the Vanguard 2 spacecraft launched in 1959 after a series of failures [6], and demonstrated optical instrumentation consisting of photocells and a telescope for measuring the intensity of sunlight reflected from cloud tops. The optical instrument successively scanned regions of the Earth using the motion of the spacecraft, like an early version of a modern push-broom imager. After the success of Vanguard 2, TIROS-1 became the first fully operational weather satellite, using a television camera for instrumentation.

After meteorology, the intelligence community took note of the enhanced capabilities that EO satellites could provide, such as full-Earth coverage without requiring aircraft to enter heavily monitored airspace. The first spy satellites were part of the CORONA program administered by the Central Intelligence Agency (CIA). The CORONA program consisted of high resolution, low field-of-view scans of particular targets, and with its pre-planned list of targets, could be considered the first "tasked" missions. These early missions stored imagery on films, with canisters deployed with a parachute back to Earth and collected by aircraft [8], as depicted in Figure 1.2. The usage of film greatly limited the maximum duration of these missions—ranging from hours to days [9], which constrained planning complexity required. Due to this inherently limited mission complexity, planning and scheduling would remain primarily manual, with targets gathered by analysts far in advance of missions [9].

The CORONA program was highly successful in its mission of surveillance, and the utility of Earth Observation to broader applications became readily apparent. Landsat-1, initially launched in 1972 as the Earth Resources Technology Satellite (ERTS) was the first EO spacecraft specifically for monitoring Earth's land mass and represented a significant advancement in instrumentation [11]. Landsat-1 contained two image instruments: the Return Beam Vidicon (RBV), a type of full-frame camera sensor, and the experimental Multispectral Scanner (MSS)

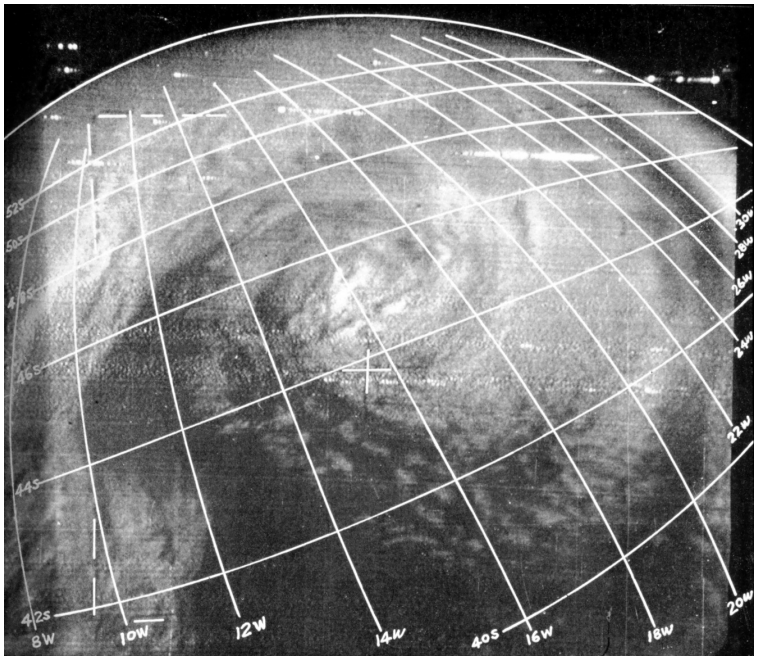


Figure 1.1: TIROS-1 image of a cyclone in the South Atlantic Ocean with overlaid latitude/-longitude lines. Taken 1960-04-28 [7].

which was the first push-broom multispectral sensor in space. The MSS was initially highly experimental, but eventually became the de facto primary instrument after on-orbit testing. Landsat-1 was the first "monitoring" mission, with the goal of taking as much imagery of the Earth's landmass as possible, subject to resource constraints. As of writing this thesis, the Landsat program is the longest running Earth Observation program, with Landsats 8 and 9 as the most recent on-orbit, and with the Landsat NEXT program in its planning phase, initially designed as a three-satellite constellation with enhanced instrumentation [12].

Steady advancement in technology through subsequent decades ameliorated many issues with spacecraft resources, which resulted in stagnation of planning algorithm development. Digital storage greatly extended satellite mission lifetimes and reduced operational complexity, and advancements in solar power technology resulted in efficiency gains [13] for monitoring missions. Some other mission types such as in the intelligence community were at that point repetitive and predictable, requiring little in operations [9, 10, 14].

1.1.2 Commercial Era (1982 - 2007)

Up until the late 1980s, all EO was operated primarily by government agencies. Whether the applications were for intelligence, meteorology, or land surface monitoring, government agencies all designed, built, and launched EO satellites. Initial policy directives in the commercialization of EO programs from the United States in the National Space Policy Directive 42 (1982) [15] and from France with the creation of Spot Image as a public-private partnership in 1982 [16] laid the groundwork for commercial involvement. The SPOT program eventually became the first commercial entity to operate EO satellites with the launch of

Figure 1.2: Recovery of a Corona film canister in mid-air from a Fairchild C-119 Flying Boxcar aircraft (1960) [10].

SPOT-1 [16]. Customers of SPOT-1 imagery included CNES, IGN, and, perhaps most famously, SPOT images were used by news agencies and governmental organizations for monitoring the Chernobyl disaster in 1986 [16].

Following success in Europe, the commercialization of EO in the 1992 Land Remote Sensing Policy Act allowed commercial EO satellite operators in the United States [17]. Subsequently, American commercial satellite operators such as DigitalGlobe emerged, offering high-resolution optical imagery to both government and private sector customers. DigitalGlobe, which later became part of Maxar Technologies, launched its WorldView series of imaging spacecraft in 2007 [18–20], offering higher resolution and more frequent global coverage with its off-nadir pointing capabilities.

Commercial operators engaged in both selling archived imagery that had already been taken, and selling future imagery requests. The decentralized source of these requests incentivized improvements in planning algorithms. Latency became an increasingly important factor—commercial operators could capitalize on frequent, short-notice requests from new industry sectors making use of EO.

These changes in the customer base along with increasing solar panel efficiency resulted in the constraint landscape for spacecraft planning and scheduling to be altered. Now, with power and onboard data less of a constraining factor, around the turn of the century, spacecraft agility became the tightest constraint to work against [14, 21, 22]. With a new class of Agile Earth-Observing Satellites, along with interest from governments, new automated planning algorithms that could take agility into account were developed in order to maximize their utility [22].

1.1.3 Modern Earth Observation (Since 2007)

Earth Observation at present is a highly advanced industry with mature, profitable commercial providers alongside government programs. Open data policies pioneered by the Landsat program with the Landsat Open Data Policy in 2008 [23] along with similar policies adopted by Sentinel-1 of the European Space Agency (ESA)'s Copernicus program further democratized access to data. An improvement in maximum resolution from 50 cm to 25 cm ground sample distance (GSD) for civilian data adopted by the U.S. Government, the inclusion of synthetic-aperture radar (SAR) in commercial EO imagery, and the availability of products such as Google Earth supported another period of dramatic growth in EO access and interest [24].

Applications for Earth observation have expanded as the democratization of EO data has allowed for new utility. Commercial uses for Earth Observation have exploded, and since frequent, high resolution commercial satellite imagery is now broadly available, there may be countless more uses that are not being publicized. Some of the largest uses for EO satellite imagery currently include:

- **Agriculture:** Multispectral and hyperspectral imagery can be used to obtain crop health, soil moisture, and vegetation indices. EO-based agricultural analytics can help improve yield through improved soil condition monitoring, and deep learning approaches have emerged that can directly estimate crop yield [25].
- **Infrastructure Monitoring:** Remote sensing data can be used for assessing construction progress, finding illegal development, and monitoring asset risk to wildfires, floods, and more, useful for both insurance and civil engineering firms [26].
- **Ship Tracking:** EO data combined with advanced analytics or machine learning algorithms can be used to detect illegal fishing, monitor port congestion, and track ships with their transponders disabled [27, 28].
- **Forestry and Land Use:** Satellite imagery has been used for illegal logging detection, biodiversity monitoring, and land cover classification [29–31].
- **Disaster Response and Humanitarian Aid:** EO data provides rapid assessments of natural disasters such as floods, wildfires, hurricanes, and earthquakes. This data has been used to identify affected areas, plan logistics, and estimate damage. Recently, EO data has drastically helped accelerate recovery efforts from the 2023 Turkey Earthquake [32].
- **Energy and Mining:** EO data supports exploration, environmental compliance, and site monitoring for oil, gas, solar, and mining industries. Imagery can be used identify potential reserves, detect flaring activity, and monitor environmental factors. Modern hyperspectral data and advanced algorithms on multispectral data can also be used for monitoring of natural gas and other leaks [33].
- **Climate and Environmental Monitoring:** EO data has long been used to validate climate models, analyze ice sheets, and estimate sea level rise. Advanced instrumentation has also started to be used to directly estimate and find carbon and natural gas emitters

around the world [34].

Additionally, open access of EO data has also resulted in unexpected use cases. A select few of these are presented below:

- **GDP monitoring:** EO-based estimates of GDP have been obtained from night-time lighting. In regimes where official data is low quality or intentionally obscured, this proxy has proven to be reliable at showing true economic activity from an unbiased estimator [35].
- **Polio vaccination:** In regions like Northern Nigeria and parts of Pakistan, satellite imagery has been used to identify previously unknown settlements. Global health organizations were able to use this data to extend polio eradication effort into these settlements [36].
- **Retail traffic estimation:** Parking lot occupancy has proven to be a useful proxy for overall activity. Hedge funds and retail analysts have used this data to estimate consumer foot traffic [37].
- **Mortgage default risk monitoring:** Satellite imagery combined with advanced analytics can be used to detect signs of neighborhood decline. Dying lawns, fewer cars in driveways, or halted construction projects can suggest increased tightening and an elevated risk of default [38].
- **Crop yield estimation:** Space-based instruments combined with advanced data analysis techniques can be used to analyze photosynthesis observations via solar-induced chlorophyll fluorescence, allowing accurate estimation of crop yields [39, 40].

As applications for Earth Observation have shifted, so too has the technology on modern Earth-Observing satellites. Incentives for latency and resolution have further cemented the role of monitoring and tasked missions. Monitoring missions with their typically wider field-of-view and always-on imaging operations could tackle low latency imagery captures at the expense of resolution, while tasked missions could tackle higher resolution areas, but often at the expense of latency, due to external input required to direct imaging operations. Examples of monitoring missions include USGS' Landsat program [41], ESA's Sentinel-2 spacecraft [42], NASA's Aqua [43] and Terra [44] spacecraft, and Planet Labs' SuperDove spacecraft. Tasked missions include Maxar's WorldView series of spacecraft, including its upcoming WorldView Legion constellation, as well as Planet Labs' SkySat constellation along with its upcoming Pelican and Tanager constellations.

Several transformative changes in the 21st century have resulted in smaller and more capable EO spacecraft:

- **Standardized form factors:** Cubesats pioneered standardized small form factors for spacecraft, allowing many different operators and manufacturers to save costs on deployment and launch. While initially adopted by academic institutions, the significant cost savings eventually found commercial usage.
- **Low-power electronics:** The mobile phone revolution of the late 2000s also resulted in advancements in the size and power efficiency of electronics. Adoption of commercial-

off-the-shelf (COTS) electronics significantly reduced satellite size, weight, and cost [45].

- **Plummeting launch costs:** The emergence of low-cost launch vehicles such as SpaceX's Falcon 9 rocket also dramatically reduced launch costs, declining from ~$30,000/kg down to ~$3,000/kg [46].

Combined, these advancements meant that small, high-performance satellites could were viable, low-cost platforms for EO, and constellations consisting of small satellites became both technologically and economically viable, increasing revisit rates, lowering latency, and increasing redundancy and resilience. Planet Labs, a commercial EO satellite operator in San Francisco, was founded upon this idea, that latency and revisit rates could be optimized with a large constellation of small satellites. Planet Labs utilized its 3U Cubesat SuperDove platform for this goal, launching a large satellite constellation spread across multiple orbital planes to minimize revisit rate. By 2020, they had launched over 350 satellites and built the largest imaging constellation in history, enabling daily global coverage of the Earth's land mass [1].

Other commercial providers also utilize distributed satellite constellations as a core part of their operations, such as:

- **BlackSky:** High revisit rate optical imagery with geospatial analytics for both defense and commercial customers [2].
- **Capella Space:** Commercial SAR imagery, unlocking imaging during night or through cloud cover, along with three dimensional representations [47].
- **Satellogic:** An EO company based in Argentina, aiming to launch over 200 satellites with high-resolution hyperspectral capabilities [3].

1.2 Future Pathways of Earth Observation and Motivation

The case has been made for the usefulness of EO data in a broad array of fields. The legacy satellite imagery pipeline may slowly fade in utility, as customers do not want EO data for the imagery, but rather *information*. However, limitations in latency and availability intrinsic to our current architecture of EO spacecraft still affect our ability to obtain low-latency information. The operational value of EO data is fundamentally time-sensitive; often its utility decreases exponentially with increasing latency between image acquisition and availability to end users. This temporal aspect presents a core challenge, where latency in information availability can critically undermine responsiveness in areas like disaster management, agriculture, defense, and intelligence to name a few.

Even with the most up-to-date technological improvements, many aspects of latency in EO have been only mitigated instead of fundamentally addressed. Even with more satellites as part of large EO constellations, linear improvements in latency will require exponentially more spacecraft in orbit, to cover more planes for higher responsiveness. EO satellites in the modern era in essence have worse responsiveness than remote control cars; the spacecraft require pre-planned schedules, have very intermittent and limited communications windows,

and have little autonomous decision-making onboard. These limitations make unpredictable phenomena challenging for EO satellites to address. Unpredictable phenomena can either serve as attractors of tasks (high-priority events to be imaged) or detractors (undesirable conditions affecting imaging tasks).

The most common detractor in Earth observation, aside from in meteorological studies, is cloud cover, which at any given time obscures approximately two-thirds of Earth's surface [4]. Cloudy data is especially a concern for spacecraft that are not a part of a constellation, due to the long revisit times involved. Task attractors include applications such as real-time wildfire tracking [48, 49], deep convective ice storm studies [50], and planetary boundary layer research [51].

Commercial satellite operators such as Planet Labs have been increasingly leveraging the information vertical, by offering their own analysis products that can fuse data from many different satellite and ground sources [1], such as soil moisture content, land surface temperature and vegetation biomass [52]. Planet Labs additionally offers a product that no longer provides *imagery* itself, but rather *insights*. Using onboard edge computing capabilities such as that on Planet's Pelican-2 spacecraft, insights can be delivered simply as latitude/longitude information along with any other accompanying metadata, which can be downlinked through a low-rate always-connected link, providing the information a customer cares about to them faster and more efficiently [53].

Dynamic tasking with body-fixed lookahead is a mission concept that sits in between conventional tasked and monitoring mission types, as a form of tip-and-cue. In this mission concept, the spacecraft has two payloads: a primary instrument for high resolution imaging capture, and a secondary payload for looking ahead and ranking upcoming imaging tasks. The two instruments ideally would have very different characteristics: the lookahead instrument would have an extremely wide field of view looking far from nadir towards the horizon, while the primary instrument is conventionally a medium to high resolution imaging system.

With modern advancements in onboard compute capabilities, including vision processing units (VPUs) and graphics processing units (GPUs), techniques such as convolutional neural networks can be used to classify and rank upcoming tasks. The spacecraft's imaging schedule can then be re-optimized based on the ranking from the vision system. Dynamic tasking in this fashion could unlock significantly improved overall satellite utilization with lower latency. Additionally, the methods have potential for significantly improving tip and cue capabilities, with the additional awareness provided by the lookahead sensor.

1.3 Thesis Layout

The layout of this thesis with links to sections is as follows:

> §1 Introduction—This chapter covers a history of Earth Observation, shows current challenges, and areas and applications in which autonomous dynamic tasking could have an outsized impact, compared to current EO scheduling techniques.

§2 Background—This chapter goes through a history of autonomous tasking missions and the algorithms they use, along with a gap analysis and how the contributions of this thesis fit in. A direct link to the gap analysis and contributions is provided here: §2.6 Gap Analysis and Thesis Contributions.

§3 Mission Concepts—This chapter goes through an array of possible dynamic tasking mission types, and outlines the exact mission architectures considered in this work: a vision-based instrument, and a proxied vision instrument utilizing data from meteorological satellites in Geostationary Earth Orbit (GEO). Additionally, this chapter discusses the exact application case used in this work: cloud cover, with two classes of spacecraft considered for analysis: an agile, and an ultra-agile case.

§5 Lookahead System Design and Drivers—This chapter presents a systems analysis of a lookahead instrument, discussing required field of view for specific modalities of tasked missions depending on parameters such as orbit altitude and field of regard. We consider two cases of lookahead instrument: one with a large enough field of view such that it can conduct stationary dynamic tasking, and a second with a narrower field of view that must re-orient for lookaheads and perform agile dynamic tasking for full observability. Methods of retrieval from meteorological satellites in GEO are also discussed.

§4 Satellite Resources, Constraints, and Scheduling—This chapter presents a deep dive into satellite resources, how they relate to satellite scheduling algorithms, scheduling algorithms used in prior work, their limitations, and the specific scheduling formulation used in this work.

§6 Sensing and Dynamic Tasking—This chapter presents dynamic tasking as an adaptive stochastic orienteering problem (ASOP), presents some background on how these are solved, and outlines the post-processing method by which dynamic tasking is implemented in this work. Heuristics for estimating dynamic tasking advantage and methods for planning lookaheads to be net-position on an expected value basis are also presented, based on a simplified version of the scheduling problem, and using a convolutional neural network with synthetic data. Results for the thesis analysis cases are presented based on each of the environments considered.

§7 Constellation Extensions—This chapter provides extensions for the dynamic tasking algorithms presented in §6 Sensing and Dynamic Tasking to satellite constellations, primarily focusing on distributing global belief state across a satellite constellation. Methods by which a compressed global state can be represented are shown and an example using a leader-follower constellation is shown where the heuristics presented in §6 Sensing and Dynamic Tasking are used with the aforementioned global state to work seamlessly across a constellation.

§8 Conclusions & Future Work—This chapter summarizes the findings from this thesis, including covering performance of dynamic tasking on each of the various considered environments. Finally, this chapter also summarizes potential avenues of future work, including alternative applications that the algorithms developed for this work could also be applicable to, as well .

1.4 Summary

This chapter recaps a brief history of EO, with initial missions meteorology and intelligence to today's commercial, constellation-based operators. We have outlined the limitations of EO, and the growing sensitivity to latency, as data has evolved from imagery, to information, and how autonomous systems can help bring more autonomy and bridge the gap between the conventional monitoring and tasked mission types. We also introduced the concept of dynamic tasking, and provided the layout of this thesis and the strategy for analyzing vision-based autonomy for EO satellites. We can now review previous work on dynamic tasking to understand limitations and challenges.

Chapter 2

Background

This chapter discusses the combination of factors necessary for implementation, looks at prior missions employing autonomy, and current dynamic tasking algorithms. Dynamic tasking involves a wide array of interdisciplinary components, so specifics on individual subsystems (such as schedulers) are given in their relevant sections.

2.1 Advancements in Computer Vision

Recent advancements in computer vision, particularly in deep learning, have resulted in high-performance vision models that can learn complex representations directly from raw L0 or L1 imagery, without requiring significant calibration or corrections, unlike traditional approaches relying on calibrated radiance or reflectance values. These models are capable of detecting features such as clouds, water bodies, or urban structures using spatial context and texture as well as spectral information. This improved vision performance allows for the use of simpler imaging instrumentation such as monochrome or RGB sensors, while still achieving high accuracy in downstream tasks [54]. Examples of these highly sophisticated models and their associated applications include cloud segmentation [55, 56], land use classification [57], wildfire sensing [58, 59], hotspot detection [60, 61], and more. High vision system accuracy and in these downstream tasks with low-power, simple and compact, auxiliary vision payloads allows for a much lower barrier to vision-based autonomy.

2.2 Compute in Space

While advancements in computer vision have made great progress, utilization of modern machine learning techniques require additional compute power. Initial autonomous technology demonstration missions struggled with available compute resources, for example, JPL'S EO-1 mission had only 4 MIPS and 128 MB available to be dedicated towards autonomy. Advancements in mobile computing and low-power electronics have crossed over into the space industry and enabled significantly increased compute capabilities.

Notable examples include ESA's OPS-SAT (2019), which incorporated an Altera FPGA with two ARM cores at 0.8 GHz [54, 62], and ESA'S Φ-Sat-1 and Φ-Sat-2 missions, which both utilized Intel's Movidius Myriad 2 vision processing unit (VPU) to enable onboard cloud detection using neural networks in a tiny 9.5×8 mm package size [63, 64]. More recently, platforms such as Capella's Sequoia spacecraft (2021) and Planet's Pelican-2 (2025) [53] have utilized NVIDIA GPUs; Sequoia is flying with a Jetson TX2, and Pelican-2 flying with a likely newer unspecified platform [53], making a significant speed up in machine learning tasks possible. Ingenuity, the Mars helicopter, utilized a Qualcomm Snapdragon 801 processor, leveraging consumer hardware for real-time vision based autonomous operations in deep space [65]. A selection of space-based computation platforms, performance, and applications is given in Table 2.1.

Table 2.1: Representative missions employing modern edge computing processors.

Mission	**Year**	**Processor**	**Compute**	**CV/ML Function**
JPL EO-1 [66]	2000	Mongoose M5	~4 MIPS	Support vector machine (SVM) science event detection and follow up, in-orbit experiments
OPS-SAT [54]	2019	Dual 800 MHz ARM Cortex-A9 + Cyclone V FPGA	~4000 MIPS	In-orbit experiments
Sequoia [67]	2020	NVIDIA Jetson TX2 GPU	~2 TFLOPS	Real-time onboard image processing
Φ-Sat-1/2 [63, 64]	2020/23	Intel Movidius Myriad 2 VPU	~ 80-150 GFLOPS	Cloud detection, vessel detection, compression
Ingenuity [68]	2021	Snapdragon 801	~2 GFLOPS	Visual navigation
CogniSAT-6 [69, 70]	2024	Intel Movidius Myriad X	~1 TOPS INT8	Real-time ship/anomaly detection
Planet Pelican-2 [53]	2025	NVIDIA Jetson Platform	1–20 TOPS	On-orbit object detection

2.3 Notable Missions and Concepts

This section covers notable missions and concepts employing autonomy techniques. The missions and concepts outlined in this section span from early onboard experiments to more recent operational dynamic tasking systems with intelligent sensing and scheduling. We make a distinction between concepts that make algorithmic improvements, and flown missions that also contribute to de-risking technology and operationalization.

2.3.1 Flown Missions

Earth Observing-1

The Earth Observing-1 (EO-1) from NASA Goddard Space Flight Center spacecraft launched in 2000 was a technology demonstration mission, hosting a large variety of new technologies including the Advanced Land Imager (ALI) developed by MIT Lincoln Laboratories, the Hyperion hyperspectral imager developed by Northrop Grumman Corporation [71], as well as phased array antennas, flexible solar panels, and atmospheric correction instrumentation [72]. EO-1 also served as the testbed for various demonstrations of onboard autonomy, including vision and perception, onboard replanning, and model-based fault detection, isolation, and recovery (FDIR) [66, 73, 74].

EO-1 was also the first spacecraft to run machine learning algorithms in space, applying a support vector machine (SVM) classifier [75] to perform onboard classification of hyperspectral imagery from the Hyperion instrument. The classifiers were used to differentiate cryospheric surface features, including snow, ice, water, land, and cloud, to detect events such as lake freezing/thawing or sea ice breakup. Due to constrained computational resources and uncalibrated onboard data, a linear SVM was used for its efficiency. A subset of 11 spectral bands from the instrument were used, as well as derived products such as all 55 pair-wise band differences and eigenvectors obtained from the training data. The SVM classifier was shown to outperform a decision tree and a threshold-based classifier, and was comparable in accuracy to an expert-designed rule-based classifier, with less development time required [75].

The Autonomous Sciencecraft Experiment (ASE) from NASA JPL, starting in 2003, performed a variety of onboard autonomy experiments on EO-1 with the goal of automating aspects of science operations onboard the spacecraft [76]. The ASE pioneered onboard re-planning and machine-learned classification for Earth observation [66, 75]. Algorithms onboard the ASE included change detection for applications such as ice breakup, cloud cover, and volcano monitoring, coupled with agile spacecraft management and planning software such as the Continuous Activity Scheduling Planning Execution and Replanner (CASPER) software [76]. With CASPER, the EO-1 could re-adjust scheduling for follow-up observations to enable enhanced revisit rates, going from once every 16 days to 5 images every 16 days [76]. One of the ASE's earliest experiments was to use onboard replanning for advanced volcano monitoring [61]. The ASE successfully detected anomalous thermal events from Mt. Erebus and autonomously re-tasked a follow-up observation, taking a capture 7 hours later with detection of volcanic activity. EO-1 also demonstrated autonomous detection of sulfur-rich Arctic springs as a mission analog for Europa [77, 78], as well as autonomous scene analysis and mapping via spectral unmixing [79].

More recently, in 2017, EO-1 was used to demonstrate onboard cloud filtering and novelty detection. Bayesian threshold and Random Decision Forests (RDFs) were use for cloud screening [73]. Additionally, novelty detection through saliency evaluation was also demonstrated [73]. While EO-1 was equipped with the versatile and highly capable Hyperion hyperspectral imager with 220 bands, ranging from 0.4 µm to 2.5 µm with a spectral resolution of approximately 10 nm [73, 76], severe computational constraints such as only 4 MIPS of free computational

power and only 128 MB of RAM [73] limited further demonstrations, despite running on a second M5 processor onboard the spacecraft. Additionally, onboard software could only access a subset and a smaller sub-region of the overall capture. These limitations precluded single-overflight planning and confined the system to follow-up observations. Despite these hardware challenges, EO-1 represented a significant breakthrough in onboard image processing and autonomy.

Intelligent Payload Experiment (IPEX)

JPL's Intelligent Payload Experiment (IPEX) Cubesat launched in 2013 was a 1U technology demonstration of parts of the Hyperspectral Infrared Imager (HyspIRI) Intelligent Payload Module (IPM) [80]. IPEX demonstrated onboard processing, data triage, salience analysis, and onboard autonomous planning. IPEX utilized ground-based scheduling via JPL's CLASP planner along with JPL's ASPEN modeling language to evaluate constraints involving resources such as data and power to produce a feasible plan to uplink to the spacecraft. The CASPER onboard planner then takes the initial schedule and, in response to inputs such as machine learning outputs on analyzed imagery and onboard resources, schedules follow-up observations or incorporates previously unscheduled, low-priority tasks [80].

IPEX contained four image processing techniques: an SVM-based image classifier, onboard spectral unmixing, a random-forest based classifier, and unsupervised salience analysis [80]. IPEX used the TextureCam image processing software to produce a segmentation output with four classes, consisting of clear surface, Earth limb or haze, clouds, and space, using a random decision forest classifier. IPEX's unsupervised visual salience algorithm was used to extract and rank subregions for data downlink triage. IPEX matured and derisked the HyspIRI IPM concept on orbit, and performed thousands of executions of its onboard autonomy software [80].

GOSAT-2

The Greenhouse gases Observing Satellite-2 (GOSAT-2) [81], launched in 2018, has the TANSO-FTS-2 (Thermal And Near-infrared Sensor for carbon Observation - Fourier Transform Spectrometer-2) as its primary instrument [82]. The instrument captures high-resolution spectroscopic measurements of greenhouse gases including carbon dioxide, carbon monoxide, and methane. GOSAT-2 as of the date of writing this thesis has the most advanced and capable operational dynamic re-tasking algorithm. GOSAT-2 utilizes an onboard programmable two-axis fold mirror in the optical train of the TANSO-FTS-2 instrument, which can perform wide adjustments of up to $\pm 40^\circ$ along-track and up to $\pm 35^\circ$ cross-track, and is primarily used to perform dynamic re-tasking via optical re-pointing the primary instrument. Its intelligent pointing mechanism initially screens the capture through a CMOS RGB video camera, executes lightweight cloud screening enabled by its wide range of bands, and then autonomously re-points the fold mirror to the least cloudy area observed. An example of the intelligent pointing mechanism in action is shown in Figure 2.1. The intelligent pointing mechanism is additionally integrated with initial observation planning, enabling both fixed grid and urban area targeting modes.

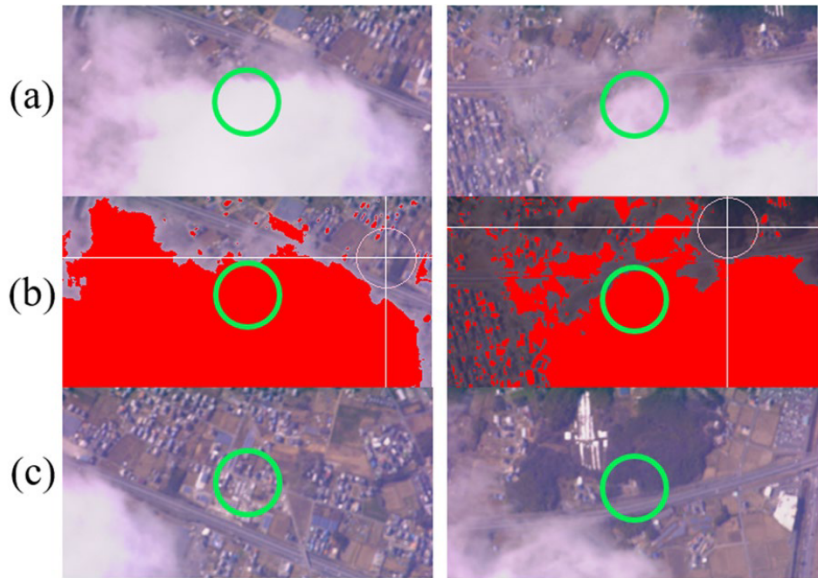


Figure 2.1: GOSAT-2 cloud avoidance for two cases (left and right), with (a) Scheduled imaging activity (b) Cloud detection and re-pointing detection and (c) Resulting image centered on cloud-free area [81]

CogniSAT-6

CogniSAT-6 is a 6U CubeSat Earth Observation mission developed by Ubotica Technologies and Open Cosmos launched in 2024 to demonstrate fully autonomous spacecraft operations [69, 70], including real-time vision-based classification of hyperspectral imagery and autonomous dynamic tasking. CogniSAT-6 integrates three primary payloads: a hyperspectral imager, an inter-satellite link (ISL) for real-time communication to mission control, and the Ubotica CogniSAT-XE2 edge processing unit, built around the Intel Movidius Myriad X Vision Processing Unit (VPU). This architecture enables onboard computer vision and inference of neural networks directly, and has been demonstrated on Lo processed data [70].

CogniSAT-6 allows for fully automated dynamic tasking. Since its edge processing is a payload and hence separate from its flight computer, significantly more of it can be utilized, in contrast with EO-1's integrated design where autonomy applications shared the main flight computer. In CogniSAT-6's concept of operations, image data is captured, georeferenced, and processed onboard to detect and geolocate relevant events. If a feature is identified with sufficient confidence, the onboard orbit propagator computes the next overpass opportunity and autonomously schedules a follow-up observation, and optionally the feature is downlinked immediately over the inter-satellite link [70].

The onboard AI pipelines rely on CNNs to classify features directly from raw Lo data, avoiding the conventional remote sensing processing pipeline. The CNN is trained directly on Sentinel-2 raw data to enable end-to-end feature detection [60]. This pipeline achieves real-time classification within 6.4 W peak power, allowing for low size, weight, and power (SWaP) onboard autonomy and dynamic tasking [60].

CogniSAT-6 now runs onboard dynamic targeting demonstrations, which utilizes its primary instrument for performing lookaheads [83], however operationalization rather than a technology demonstration would likely use a dedicated wide field-of-view lookahead instrument perhaps of a different modality to ameliorate resource costs of slewing for lookaheads [84].

2.3.2 Concepts

OPS-SAT

OPS-SAT was a mission from the European Space Agency (ESA) launched in 2019 aiming to provide an "in-space laboratory" platform where experimenters from all around the world can de-risk mission concepts and software by demonstrating them on OPS-SAT first. OPS-SAT hosts a wide variety of different hardware components that are available for experimenters to utilize, including an Altera Cyclone V FPGA with a dual-core 800 ARM Cortex-A9 CPU and 1 GB of DDR3 flash. OPS-SAT also has a 5 MP RGB camera with an approximate ground sample distance (GSD) of 80 m [85].

OPS-SAT provides the ability to rapidly iterate on algorithms in space due to a combination of the highly mature and automated operations provided by the European Space Operations Centre (ESOC), as well as the versatile high-bandwidth radio, allowing even large experiments up to approximately a 10 MB limit to be uploaded to the spacecraft. For this reason, OPS-SAT has been popular for demonstrations of computer vision enabled operations for spacecraft [54, 62]. Additionally, containerization, deployment and other systems considerations of dynamic tasking and running machine learning algorithms onboard the spacecraft have been designed and developed on OPS-SAT [86].

Machine learning based image processing algorithms have been demonstrated on OPS-SAT, ranging from extremely simple thresholding methods up to deep learning methods [54], and from applications between classification, segmentation, and clustering [54, 62]. While OPS-SAT does not have a dedicated lookahead sensor, its single camera has been explored in simulated scenarios, using the top half of its camera as a simulated virtual lookahead sensor [87]. OPS-SAT demonstrated huge advancements in onboard vision and machine learning systems, but unfortunately it de-orbited before missions demonstrating autonomous dynamic tasking could be flown.

DLR FireBIRD

FireBIRD was a mission led by the Deutsches Zentrum für Luft- und Raumfahrt (German Aerospace Center, DLR) aiming to demonstrate and operationalize high-resolution thermal infrared (TIR) remote sensing for wildfire and thermal hotspot monitoring.

FireBIRD consisted of two spacecraft: TET-1 (Technologie-Erprobungsträger 1;Technology Experiment Carrier 1) and BIROS (Bi-spectral InfraRed Optical System) launched in 2012 and 2016 respectively. The satellites were equipped with uncooled microbolometer-based thermal infrared sensors operating in the 8 µm to 14 µm spectral range, paired with auxiliary visible and near-infrared (VNIR) imagers, providing simultaneous multi-spectral imaging capability, with the goal of allowing for both contextual scene interpretation and detailed

thermal analysis. The thermal instrument has a comparable ground sample distance with conventional TIR instruments such as those on Landsat 8 [88], with a GSD of 160 m [48, 49].

Dynamic tasking for FireBIRD was proposed and investigated, but ultimately was not demonstrated on-orbit. The proposals were for two VAMOS (Verification of Autonomous Mission-planning Onboard a Spacecraft) experiment modules. OBoTiS, an onboard timeline selector that would activate pre-optimized timeline fragments in real time subject to resource constraints, and OBETTE, an onboard scheduler capable of generating additional schedule commands when the BIROS' Hot-Spot Recognition Sensor or the VIMOS cloud classifier detected a fire signature, or when data arrived through the onboard OrbComm modem [49].

The new commands would have inherited priority, resource bounds and pre-optimized templates. OBoTiS would then validate them against the current spacecraft state to avoid overruns or overlaps, and select fragments based on a greedy algorithm [48]. While simulations show that the dynamic tasking algorithms that were a part of FireBIRD would result in faster response times and higher duty-cycle utilization of the spacecraft, the software was never fully operationalized or activated beyond initial integration tests. However, the timeline fragmenting approach represented a significant advancement in dynamic tasking [49].

2.4 Dynamic Targeting Algorithms

In this work we make a distinction between dynamic *targeting* and dynamic *tasking*. We consider dynamic *targeting* as the ability to acquire new information about the upcoming orbit track, identify areas of interest, and re-optimize and execute the spacecraft's onboard schedule, in a single overflight. We consider dynamic *tasking* to be this same concept but where there is an initial pre-optimized schedule obtained from a separate ground scheduler, which is then modified onboard the spacecraft.

This definition does not set any particular constraint on the spacecraft configuration and perception system. Therefore, it encompasses various mission types, including scenarios where the spacecraft is equipped with a dedicated sensor, uses its primary instrument as a lookahead sensor, is part of a leader-follower setup with the leader possessing a lookahead sensor, employs geostationary satellites as a virtual lookahead, or integrates a sensor web that combines data from multiple terrestrial and space-based sources [89].

Dynamic targeting and onboard autonomy are certainly not new concepts. The Autonomous Sciencecraft Experiment demonstrated on JPL's EO-1 spacecraft in 2003 was one of the first to do onboard re-planning but did not have enough computational power for single-overflight planning and hence mainly looked at followup observations [76, 90, 91]. Elements of dynamic targeting were proposed by Damiani [21], where a continuously operating anytime planner would take input from a forward-looking observation instrument and re-plan a schedule of tasks, transmitting to a constellation through inter-satellite links. Lenzen [49] proposed a resource-aware onboard planner for DLR's FireBird mission consisting of two spacecraft TET-1 and BIROS, which would scan using an infrared camera upcoming regions and schedule follow-up observations of high-temperature events looking for forest fires, assembling scheduling fragments based on resource bounds, as discussed in §2.3.2 DLR FireBIRD. Heuristics for

lookahead pointing using a spacecraft's primary instrument were developed by Chien and Troesch [92], defining possible search and follow-up viewing angles for dynamic tasking. Dynamic targeting is more similar to a set covering problem as we wish to have as much lookahead of upcoming tasks while expending the least amount of resources, which are limited and shared between lookahead and imaging. Integrating this problem directly into a global scheduler can become computationally challenging, and pointing heuristics like these allow for abstracting some of the problem [91, 92]. Other strategies include approaches localized to the spacecraft [93] with extensions for constellation-wide awareness [94], and decentralized and federated scheduling techniques [95]. Additional research includes looking at dynamic targeting algorithm comparisons [96]. Application cases for dynamic targeting include deep convective ice storm research [84, 97, 98], planetary boundary layer research [51], and spectral search applications [69, 70].

2.5 Alternatives to Body-Fixed Lookahead

Dynamic targeting with body-fixed lookahead is only one mission concept that tackles the problem of unpredictable phenomena in Earth observation. This section discusses some alternative concepts that are either completely remove the idea of a lookahead sensor, or are extensions or relaxations to the idea of lookahead.

The simplest solution for onboard autonomy is to screen and rank images already taken by the spacecraft's primary imager and filter data before downlink [54, 99, 100]. Data volume scheduling has been applied to Earth-observing spacecraft [101], as well as to generalized solar system bodies [102].

Sensor webs externalize the sensing to alternative space or ground sensors. This broadly specified technique allows for fusion of diverse data sources which can be made extremely robust for specific applications. Sensor webs have been developed and successfully operationalized [103], being used for volcano monitoring [104], flood monitoring [105], and more.

Significant investment in meteorological satellites by the United States, European Union, and Japan has resulted in meteorological satellites in Geostationary Earth Orbit (GEO) covering the vast majority of Earth's landmass as shown in Figure 5.9. Since these satellites are dedicated to meteorology, they image in bands that are extremely amenable for cloud detection, such as the mid-wave infrared (MWIR) and thermal infrared (TIR). Using meteorological data can be considered a form of sensor web, with a specific subset of in-space sensors with extremely wide coverage and applicability to many phenomena.

2.6 Gap Analysis and Thesis Contributions

Comparing previous literature on dynamic tasking reveals several potential gaps. Damiani [21], Lenzen [49], and Nag [94] all create formulations for onboard scheduling ability. However, the integration of lookahead sensors varies, with Damiani [21] and Candela [84] utilizing steered instruments with assumptions that the lookahead instrument can observe a fixed window in its track. Pointing heuristics are primarily researched by Chien [92] and

used as part of the Autonomous Sciencecraft Experiment on EO-1 [91]. Previous work from Nag [94] and [21] extend these onboard scheduling concepts to constellation scheduling, but only Damiani [21] considers the problem of constellation-wide lookahead, simplifying it by assuming always-available inter-satellite links. The problem of constellation-wide data downlink and replanning has been explored by Kennedy and Dahl [106, 107]. Hence, there is a gap in dynamic tasking with body-fixed lookahead with extensions for satellite constellations.

Table 2.2: Comparative analysis of features of previous onboard scheduling work, focusing on integration of a ground scheduler, lookahead methodology, and ability to extend to satellite constellations.

Work	**Methods**				
	Onboard Scheduler	**Ground Scheduler**	**Lookahead Sensor**	**Lookahead Heuristic**	**Constellation Extensions**
Damiani (2005) [21]	✓	✓	Windowed	✗	✓
Lenzen (2014) [49]	✓	✓	✓	✗	✗
Chien (2015) [92]	✓	✗	✗	✓	✗
Nag (2019) [94]	✓	✓	✗	✗	✓
Candela (2022) [84]	✓	✗	Windowed	✗	✗
Kacker (2025)	✓	✓	✓	✓	✓

The contributions outlined in this thesis are as follows:

1. **Software Package for Dynamic Tasking Simulation and Analysis:** An extensible and flexible software package has been developed to support this research, as no existing software has the breadth to support all the different required components. More information can be found in §6.3 Software Requirements and full code and documentation can be found at https://github.com/Shreeyam/phd_code, as well as source code for core components in §A Core Implementation Details.
2. **Analytic and Learned Heuristics for Lookahead Utility Estimation:** Metrics for quantifying the utility of a lookahead action as well as heuristics to estimate the utility of a lookahead action have been developed, both from first principles from an approximate scheduling problem and from data, using machine learning. An analysis of agile dynamic tasking with these heuristics as well as using just-in-time commanding with meteorological data compared with conventional EO scheduling performance is also conducted. This contribution is the primary contribution of this thesis.
3. **Constellation Extensions with Shared Task States:** Extensions of the heuristics as listed in Contribution 2 to allow them to work across a constellation with awareness channeled through shared access belief states.

2.7 Summary

This chapter reviewed the foundations of vision-based autonomy, and the combination of advances in computer vision and onboard computing that now make advanced onboard vision-based autonomy feasible. We have also reviewed previous breakthrough missions serving as technology demonstrators for onboard autonomy, as well as concepts that did not get demonstrated. From these, we also analyzed dynamic tasking literature, analyzed gaps, and introduced the main contributions of this thesis. We can now analyze dynamic tasking subtypes, and evaluate in detail the constraints of operationalizing dynamic tasking on current EO satellites.

Chapter 3

Mission Concepts

As mentioned in §2 Background, dynamic tasking can refer to a broad range of different kinds of mission, so long as perception and response within a single overflight occur. This chapter expands on dynamic tasking formulations, their interconnections, and also motivates utilizing onboard vision or proxy vision instruments, such as through other spacecraft. This chapter also introduces the analysis cases in the rest of this thesis.

3.1 Concept of Operations

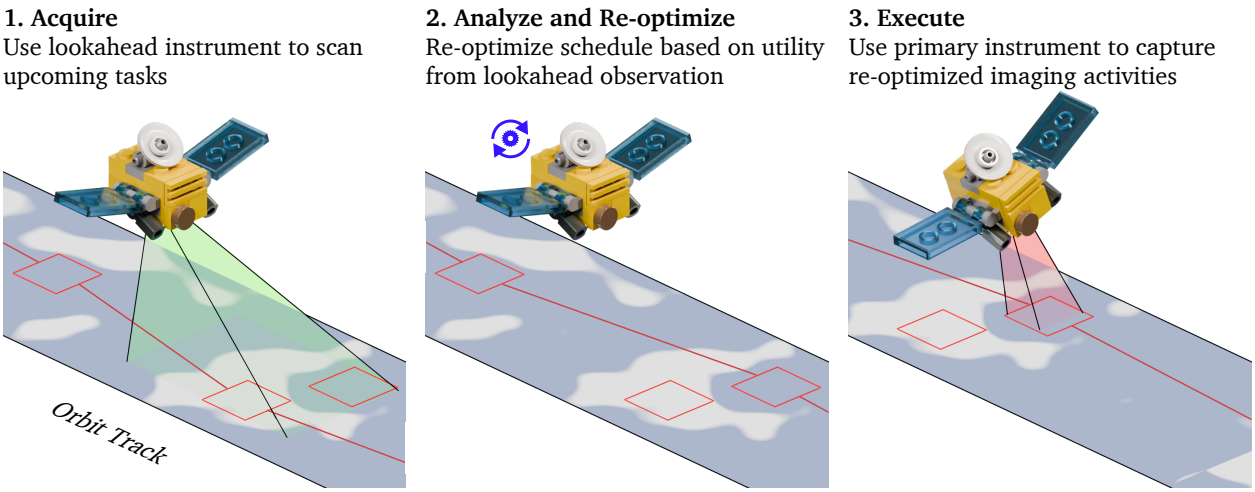


Figure 3.1: Dynamic tasking with body-fixed lookahead framework with initial acquisition, analysis, and re-optimization of schedule onboard spacecraft.

Dynamic tasking in this work is defined as the capability for a spacecraft to acquire, analyze, and re-optimize information about upcoming image accesses within the span of a single overflight, as shown in Figure 3.1. As mentioned in §2.4 Dynamic Targeting Algorithms, we consider dynamic *tasking* separate from dynamic *targeting*, incorporating additionally an initial schedule solution to be modified.

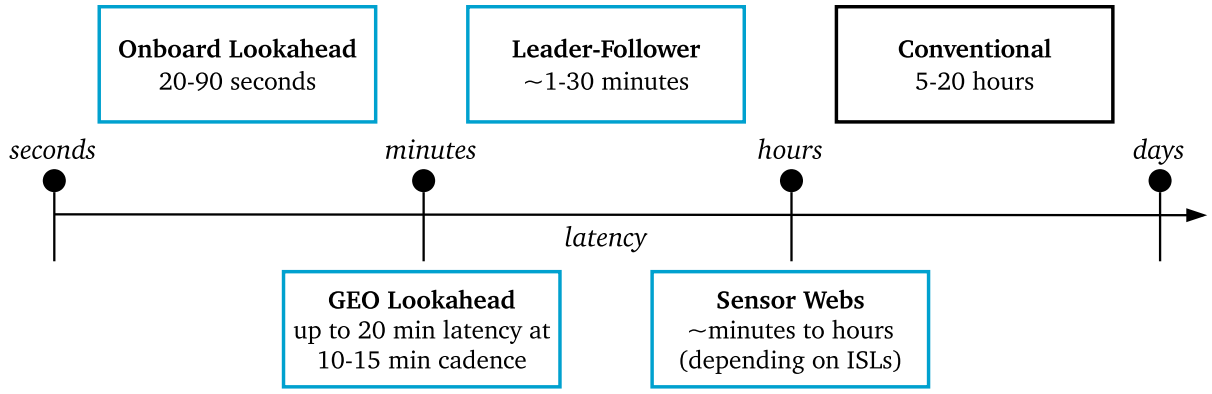


Figure 3.2: Overview of different dynamic tasking mission concepts and associated information latency for each one, in comparison to conventional scheduling methods.

Many different types of sensors and spacecraft configurations can be used for lookahead and image capture, with varying amounts of latency between them. Figure 3.2 shows examples of different dynamically tasked mission types with a comparison to a centralized ground-based scheduler. Using an onboard lookahead sensor in LEO allows information latency ranging between approximately 20-90 seconds, and has the lowest latency due to viewing in the immediate vicinity of the spacecraft. The next lowest latency is likely in a leader-follower constellation, where the belief state between spacecraft can be shared, to avoid wasting effort—in this situation, the leader can do lookaheads and some imaging, while the follower gains most of the benefits of the lookahead without actually doing them, subject to task utility not changing rapidly enough during the gap between leader and follower that the information performs on par with chance. Utilizing a networked sensor web can have a wide range in information latency, as specific architectures for the exact sensors, networking latency, and tasking latency can create huge variations in how fast updated information can get to the spacecraft [103–105, 108, 109]. Using meteorological satellites in Geostationary Earth Orbit (GEO) is one form of sensor web, where the information useful to planning upcoming tasks is transferred from GEO to an imaging spacecraft in LEO using either a bent pipe relay or inter-satellite link. Using this architecture, the minimum information latency considering the meteorological downlink pipeline alone can be as little as minutes, with up to a 10 minute cadence [110]. In this work we look at two cases: (1) **a primary case looking at using a body-fixed lookahead sensor**, separate from the main imaging instrument, and (2) **a secondary case utilizing real-time data from meteorological satellites** for autonomous tasking. While spectral properties, resolution, and other aspects of imager performance impact dynamic tasking, they add complexity for analysis in terms of modeling and data availability, and are extremely application specific. Here, all aspects of the imager properties and vision model are abstracted: if an access is visible in the camera's projection, the access utility can be obtained exactly.

Considering the primary case of a body-fixed lookahead sensor, most utility is derived by maximizing the view of all upcoming tasks within the horizon of the spacecraft, subject to resolution constraints of whatever vision based model is used to obtain updated utility values. Hence for a body-fixed sensor to be used effectively for dynamic tasking, desirable

qualities include a wide field of view with the boresight angled up towards the horizon. This configuration allows the spacecraft to perform a lookahead prior to reaching a scheduled set of targets, with the option of reorienting the spacecraft to perform a lookahead if doing so is expected to yield greater utility, acknowledging that such reorientation incurs costs in terms of both resources and time.

In the secondary case of utilizing meteorological data, considerations include spectral bands available on the instruments, the cadence at which information can come in, if there is an ISL available or if all information is sent through ground stations, and how much information latency is present in the link. Significantly less systems analysis and engineering is required, primarily because because this operating mode depends on a few select meteorological satellites that are already in space.

Following a lookahead capture, onboard systems must convert image pixels into georeferenced world coordinates. Additionally, some form of vision model must also run to derive updated access utilities from the lookahead capture. Accurate timing, orbit determination, attitude estimation, and sufficient onboard compute resources are essential for this process. Based on the updated information, utility values for candidate tasks can be reassessed, and the schedule may be modified.

Schedule re-optimization can have a large variation in complexity. In the simplest case, tasks below a specific threshold utility can be excluded from the schedule. Increasing in complexity, task "fragments" can be swapped in and out of the schedule, similar to algorithms for finding maximum independent sets [111, 112]. In the full re-optimization case, the spacecraft also hosts onboard a list of accesses that function as alternates for the schedule. A lightweight optimizer can then be used to perform a local optimization of the spacecraft schedule, essentially performing a receding-horizon optimization, as the spacecraft executes more and more of the schedule. If needed, the spacecraft can also broadcast its updated belief state across a satellite constellation, which can be used to inform the rest of the satellite constellation about new estimates for observed task utilities, mark off the task in the current schedule as observed, or facilitate autonomous tip-and-cue across a constellation. Updating task utilities in this format can especially help in the case of a heterogeneous constellation, e.g. a bulk of imaging satellites for conventional imaging tasks, along with a small number synthetic aperture radar (SAR) constellation to capture high value imaging accesses that could not be imaged due to cloud cover [113].

3.2 Image Request Dataset

Previous work on spacecraft scheduling has either not disclosed their datasets of requests, use proprietary data [114], or uses a proxy dataset. Proxy datasets include random samplings of the Landsat grid [115], and the largest 10,000 world cities [14, 116]. Figure 3.4 shows the request volume from commercial customers of Planet Labs high resolution imagery. Qualitatively with comparison to Figure 3.3 it can be observed that most imaging demand comes from areas where people live, with a bias towards areas of higher economic activity. To most closely mirror commercial spacecraft requests, this work will consider the top 10,000

largest world cities as its primary dataset. Using a stationary set of requests also allows for abstracting where they come from: so long as the requests are in $\mathcal{R}$, it does not matter if the requests are obtained from a single customer, or aggregated from many different ones.

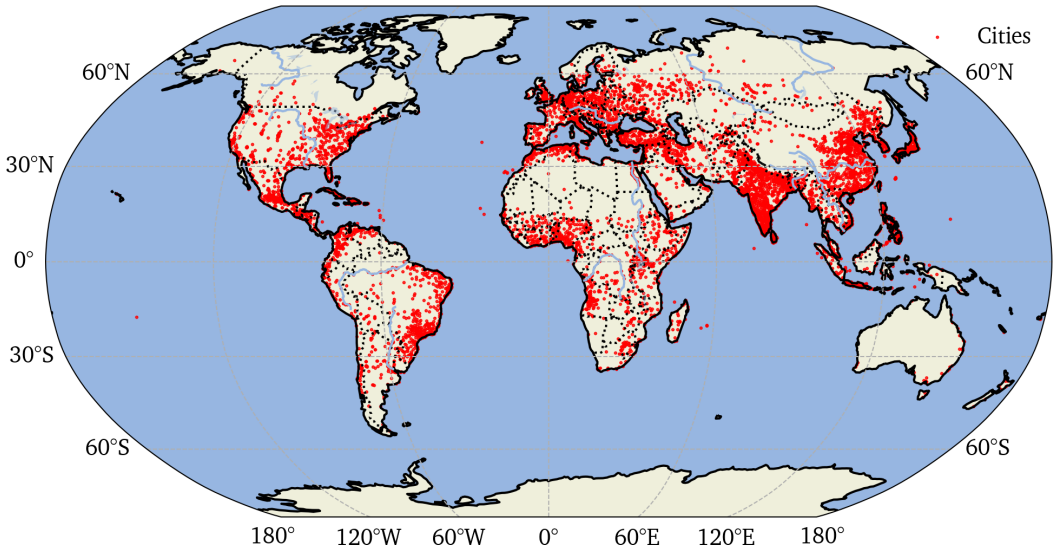


Figure 3.3: Map of 10,000 largest world cities used as proxy for satellite request locations [116].

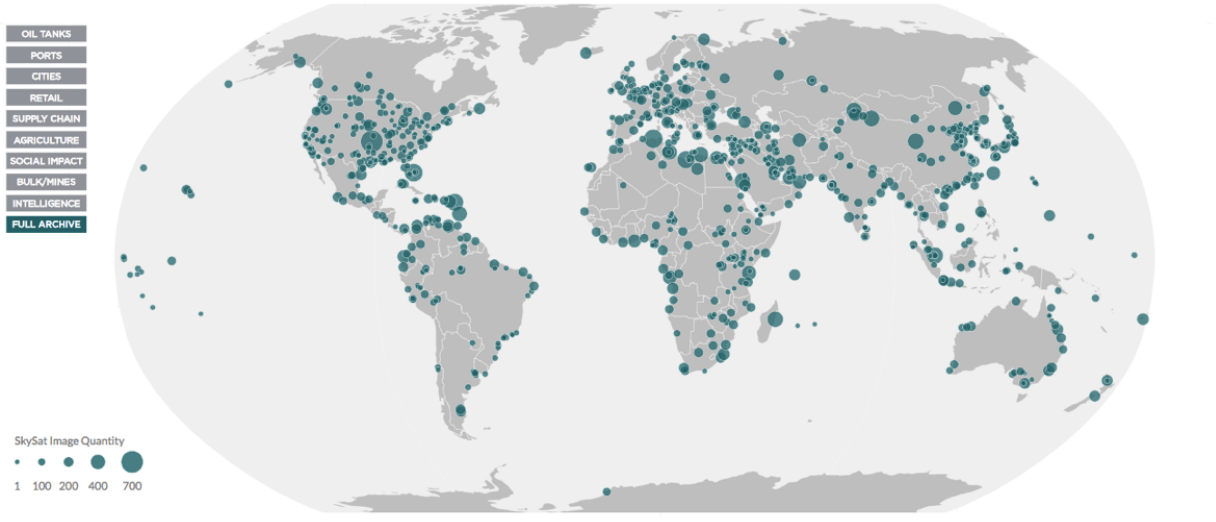


Figure 3.4: Planet SkySat high resolution collection map as of September 2017 [117]. This map is qualitatively similar to the one shown in Figure 3.3, although with an additional economic component.

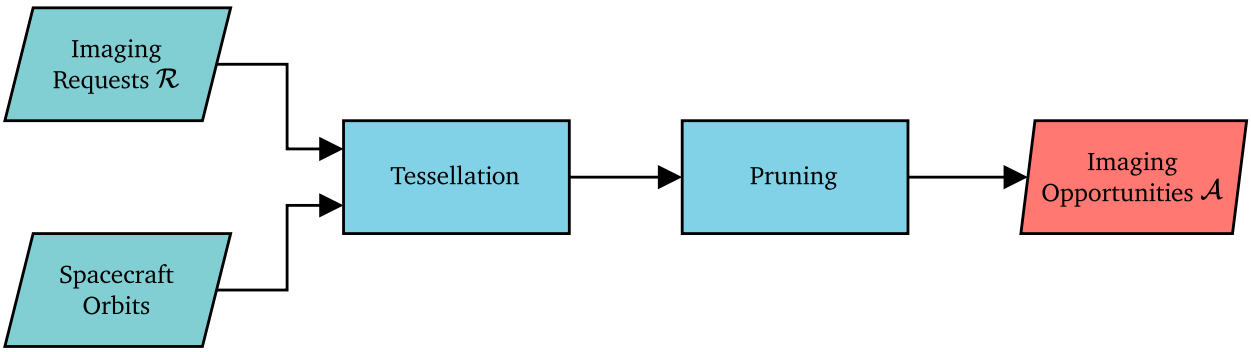


Figure 3.5: Flow of converting imaging requests from customers into specific imaging opportunities [14, 118]. Image requests are decomposed into point collects by a tessellator, then pruned, before collected into all imaging opportunities (accesses) $\mathcal{A}$.

3.3 Spacecraft Scheduling Pipeline

The spacecraft imaging pipeline first starts with imaging requests. Depending on the type of Earth-observing spacecraft, these requests may originate from different sources. Military satellites may collate requests based on the needs of the intelligence community alone, while commercial Earth-observing satellites such as those from Planet Labs and Maxar will gather requests from customers around the world. In this work, we consider primarily the applications of commercial spacecraft because data about their operations is more readily available.

Imaging requests can be grouped into four categories: point, strip, and area collects. Point collects refer to the capture of imagery at specific, discrete locations, often used for high-priority or high-interest areas. Strip collects involve capturing imagery along a continuous path, which is particularly well suited for spacecraft with push-broom type imaging sensors. Area collects refer to collections that require inclusive coverage of a predefined area, conventionally given as a polygon [14].

These collection types need to be decomposed in a format that is amenable to spacecraft instruments. Spacecraft sensors for Earth observation are typically either full-frame sensors or push-broom sensors arranged in an array [119, 120]. With no loss of generality, we can decompose all of the request types into point collects. This is trivial for all collection types listed except for strip collects, which can be decomposed to the scheduler as two point collects at the start and end of the strip, with the constraint that they must be sequential and required elevated power and data so that the entire strip can be collected continuously between the start and end.

A tessellator, which converts the spacecraft's ground footprint into tiled interlocking sections, can be used to aggregate imaging requests into point collects with the aforementioned caveat that strip collects for push-broom sensors are decomposed into two points with additional constraints. Using the point-decomposed imaging requests with an orbit propagator helps convert each imaging request in $\mathcal{R}$ into a set of imaging opportunities, otherwise known as accesses $\mathcal{A}$, which includes information such as overpass time and roll angle. More details on the implementation of the access aggregation algorithm used in this work can be found in

§6.2.2 Access Aggregation.

Tasks can be aggregated together if the resulting grouping is more efficient from scheduling perspective. A practical heuristic for if tasks should be aggregated is if the resources required to get from one task to another are less than that required to go to the "home" position of the spacecraft, which is typically specified as either nadir or with solar panels pointing towards the sun. This approach not only saves scheduling time but also conserves satellite resources.

Imaging requests are considered "missed" if the imaging opportunity was never taken, or if some aspect of the collected image is not suitable for use. For commercial spacecraft operators this could mean that a particular image is not billable to a customer. While there is more literature on conventional spacecraft scheduling, there is little publicly available information on what classifies a task to be missed. For the purposes of this work the missed task criteria are abstracted as a binary variable, which can be extended in cases where the exact criteria are known.

3.4 Image Utility Model

The utility of satellite imagery is strongly long-tailed: a small fraction of accesses can yield significantly higher utility, while the vast majority provide routine or redundant data. A convenient way to capture this effect is with the Pareto distribution: long-tailed distributions naturally model domains where extreme outcomes, though infrequent, dominate the expected return and align with empirical observations. For example: use cases such as disaster response, activity monitoring, and domain awareness.

- **Disaster response:** A single cloud-free overpass immediately after an earthquake contributes the majority of response value, while dozens of routine passes have otherwise marginal utility.
- **Activity monitoring:** Observing a "dark" fishing vessel with no automated identification system (AIS) signal may provide more utility than monitoring compliant traffic.
- **Domain awareness:** Identifying an anomalous soil moisture reading or a surprise stockpile shift in a quarry or forest can outweigh hundreds of otherwise predictable observations.

Formally, we use two approaches for modeling the intrinsic (access-level) utility c_i. We use one approach where all accesses are worth equal amounts

$$c_i = 1, \forall i, \tag{3.1}$$

and another using a long-tailed Pareto distribution

$$c \sim \text{Pareto}(\kappa, c_{\min}), \tag{3.2}$$

where $c_{\min} > 0$ is a floor on actionable utility and $0 < \kappa < 2$ reflects how much the tail

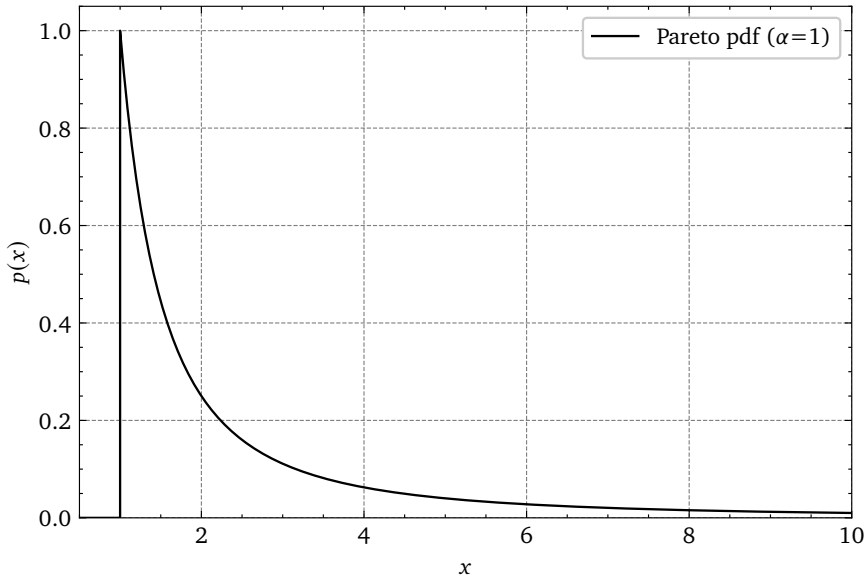


Figure 3.6: Pareto distribution in the case where $\kappa=1$, $c_{\min}=1$. A Pareto distribution is long-tailed, meaning that many samples are worth near the minimum amount, but the right tail spans to infinity, hence there is a small probability of drawing samples worth significantly larger amounts.

governs overall value. In this work, $c_{\min}$ and κ are both fixed to unit value. An example of this particular distribution is shown in Figure 3.6.

We treat the true utility of an access as modulated by current cloudy conditions, treating a point access as zero utility if it is cloudy, and its intrinsic, full utility otherwise. Hence, the utility of an imaging capture can only be realized when the target is observable. We therefore modulate c by a deterministic factor that captures sky conditions: In practice, the value of an access may be modulated by more factors, such as off-nadir roll angle due to orthorectification requirements, but in this work th

$$u_i = c_i \cdot \mathbf{1}_{\{\text{clear}(r_i)\}}, \tag{3.3}$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function and $\text{clear}(\mathbf{x}, t)$ is true when the target latitude–longitude $\mathbf{x}$ is cloud-free at time t. Thus,

$$u_i = \begin{cases} c_i, & \text{if the target is cloud-free,} \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

3.5 Metrics and Research Questions

Based on the the research gaps as described in §2.6 Gap Analysis and Thesis Contributions, we can arrive at two natural research questions around dynamic tasking:

Q1. **How do we combine lookahead actions with the existing scheduler?**

Q2. **How much better can we do compared to without dynamic tasking?**

We can use the following metrics to evaluate our new dynamic tasking performance, in the case where each task has equal intrinsic utility [14]:

M1. **Total number of collected captures:** $|\sigma|$

M2. **Total number of collected clear captures**: $|\sigma_{\text{clear}}|$

M3. **Percentage of non-cloudy imagery:** $\frac{|\sigma_{\text{clear}}|}{|\sigma|}$

This scenario compares raw schedule improvement, without additional noise incurred by the distribution of task utility. Additionally, these can be plotted on a 2D graph and compared against the following cases:

C1. **Conventional:** using no dynamic tasking data and implicitly assuming all accesses are cloud-free, even though they may not be.

C2. **Omniscient:** a theoretical case where perfect knowledge of the future is given in the simulation to assess the optimal (in the case of a MILP scheduler) schedule performance.

C3. **Dynamically tasked:** utilizing algorithms in this work to improve on the conventional schedule.

The conventional scheduling strategy, which attempts every opportunity implicitly assuming all of them are clear, produces the longest schedule; it effectively samples Earth's intrinsic cloud distribution and therefore yields the lowest fraction of cloud-free images [4]. At the opposite extreme, the omniscient strategy with perfect knowledge of future task states selects only cloud-free accesses. Its plan is shorter yet achieves 100% cloud-free acquisitions, providing a hard upper bound on cloud-free capture performance.

Finally, to make the results more applicable, we also consider cases using a Pareto-distributed utility function. These cases can provide numbers that may be more applicable to operation based on actual customer requests. The metrics in this case are modified to be as follows:

M4. **Total sum of capture utility:** $J(\sigma) \coloneqq \sum_{i \in \sigma} u_i$

M5. **Percentage of non-cloudy imagery:** $\frac{|\sigma_{\text{clear}}|}{|\sigma|}$

The gap between all the cases is likely to grow significantly in the case of Pareto-distributed tasks, as the long-tail distribution allows for capture of a single high-value task to completely alter the utility of the entire schedule. Statistically analysis is therefore difficult, and so these cases are presented using actual weather data as examples of specific operationalized dynamic tasking cases rather than for drawing overarching conclusions about algorithms.

3.6 Application Cases

This work is primarily motivated by application cases relevant to commercial Earth observation operators. Hence, application cases are selected adjacent to operational characteristics of existing commercial Earth-observing spacecraft platforms. Table 3.1 shows a variety of currently operational EO spacecraft and information about their publicly available operating characteristics. Within the satellites in Table 3.1, 45° is the most common off-nadir pointing angle, and has hence been adopted as the primary off-nadir pointing angle in this work, which also sets the field of regard to be the cross-track distance that lies within that off-nadir angle.

Agility characteristics are even more challenging to obtain from commercial satellite operators due to limited publicly available data. Metrics such as settling time and slew duration are often considered proprietary and may vary significantly across spacecraft iterations. However, we consider agility the most important quantity, as it has the most impact on the feasibility and success of dynamic tasking, due to often being the tightest scheduling constraint. For example, in the limiting case of infinite agility (i.e. all slews are instant), the system becomes effectively memory-less, no longer depending on or constrained by any prior action or state. This memory-less property implies that every access that can be obtained can also be observed, removing the oversubscribed nature of the problem. This property eliminates the need for complex scheduling, and reducing dynamic tasking to a near-instantaneous decision problem—accesses only need to be observed an instant before they are imaged to be able to remove cloudy ones from the schedule. On the other hand, a certain baseline level of agility is required for dynamic tasking to be useful. Low-agility platforms can render dynamic ineffective—for example, in the case where slew times exceed the duration during which a target transitions from the horizon to nadir, dynamic tasking would be wholly ineffective.

Table 3.1: Field of regard and pointing capabilities of selected tasked Earth-observing satellites, for targeted instruments generally imaging near-nadir.

Satellite	**Swath Width [km]**	**Off-Nadir Pointing θ_{FoR} [deg]**
SPOT-6 [121]	60	±45°
WorldView-3 [122]	13.1	±45°
Pleiades-1A/1B [123]	20	±47°
GeoEye-1 [124]	15.2	±60°
IKONOS [125]	11.3	±45°

Table 3.2 shows publicly available data on agility specifications. Since there is a mix of formats available, the available data has also been rescaled to that of a 90° slew, to make the slew time applicable to that of a full slew across the ±45° field of regard chosen in this work. In cases where the provided specification is only listed as a time, a linear scaling is used, whereas in cases where the provided specification is an acceleration and a max slew rate, a bang-bang [128] slew at max acceleration is used.

Dynamic tasking and agility are tightly coupled: dynamic tasking can only meaningfully improve performance on agile spacecraft platforms that can accommodate rapid re-targeting.

Table 3.2: Agility specifications for various satellite platforms, scaled to a 90° slew. Scaling is done linearly in the case where acceleration specifics are not given, and full bang-bang specifications otherwise. All missions listed are tasked EO satellites.

Satellite Platform	Launch	Agility Description	Scaled 90° Slew [s]
Pléiades 1A/1B [123]	2011	60° in 25 s (includes settling)	37.5
Pléiades Neo [126]	2022	60° in 20 s	30.0
WorldView 1 [19, 127]	2007	2.5 $^\circ\,s^{-2}$ accel; 4.5 $^\circ\,s^{-1}$ max rate	24.7
SPOT-6 [121]	2012	30° in 14 s	42.0

Based on the agility performance in Table 3.2, two agility cases are considered in this work: (1) the case of an **agile satellite**, which pushes the limits of what can be achieved today, to show the potential benefits of dynamic tasking on existing bus designs, and (2) that of an **ultra-agile spacecraft**, to show what might be possible in the future. These cases are shown in Table 3.3. As aforementioned, dynamic tasking requires a high agility bus to be worthwhile. For this reason, a comparison case for a spacecraft bus with more conventional agility characteristics has been omitted. Further discussion of agility's impact on scheduling can be found in §4.1.3 Agility. While agility is a prime driver when considering dynamic tasking with a body-fixed lookahead sensor, an electronically steerable instrument such as the fold mirror on GOSAT-2's TANSO-FTS-2 spectrometer [81] or a radar-based instrument [129] decouple spacecraft agility from lookahead actions and ameliorate agility resources constraints associated with agile lookahead.

Table 3.3: Slew and settling characteristics by agility class, for comparison cases used in this thesis.

Class	Field of Regard	Rotating Time (across FoR)	Settling Time
Ultra-agile	45°	15 s	5 s
Agile	45°	25 s	10 s

3.7 Summary

This chapter introduced various forms of dynamic tasking, narrowing our focus to using meteorological data and body-fixed lookahead. Additionally, the spacecraft scheduling pipeline, along with accompanying datasets and metrics used in this work have also been introduced, and, following an analysis of current state-of-the-art spacecraft performance, we select an agile and an ultra-agile spacecraft case to analyze. Now, we can evaluate the scheduling problem in detail and inspect how various operational constraints affect scheduling performance.

Chapter 4

Satellite Resources, Constraints, and Scheduling

Scheduling algorithms for Earth-observing spacecraft are useful to cover as a starting point for dynamic tasking algorithms, as building incrementally on existing infrastructure that already services space missions drastically increases the applicability of the algorithms developed in this work, making the requires changes incremental instead of starting afresh. This chapter primarily focuses on scheduling from a centralized scheduling system rather than a distributed or federated scheduling system, as this is the simplest case.

4.1 Satellite Resources

Earth-observing spacecraft have various resource constraints that must be managed carefully during operations. The main resources that must be considered for scheduling problems are:

1. **Power**: The availability of power from the batteries and solar panels.
2. **Data**: The availability of onboard storage and downlink bandwidth.
3. **Agility**: The ability of the spacecraft to re-orient itself and settle.

Certainly this is not an exhaustive list of *all* spacecraft resources that must be considered during operations but are the ones that are most likely to be constraining during day-to-day operations of imaging spacecraft.

Modeling all of these resources at high fidelity can become computationally intractable when considering the combinatorial nature of scheduling problems [130]. Spacecraft resource management is its own field of study due to the inherent complexity of managing a fleet of assets that cannot be serviced in a hostile environment. Modeling all aspects of the these resources in high fidelity often requires non-linear non-continuous optimization with lots of parameters to be accurately modeled [130]. For scheduling problems, simplifications are often made to these resources in order to make them computationally tractable for the scheduler

to handle. The key thing for the simplified constraints is that they must over-estimate the required resources for actions so that some margin is built in during execution of the schedule.

4.1.1 Power

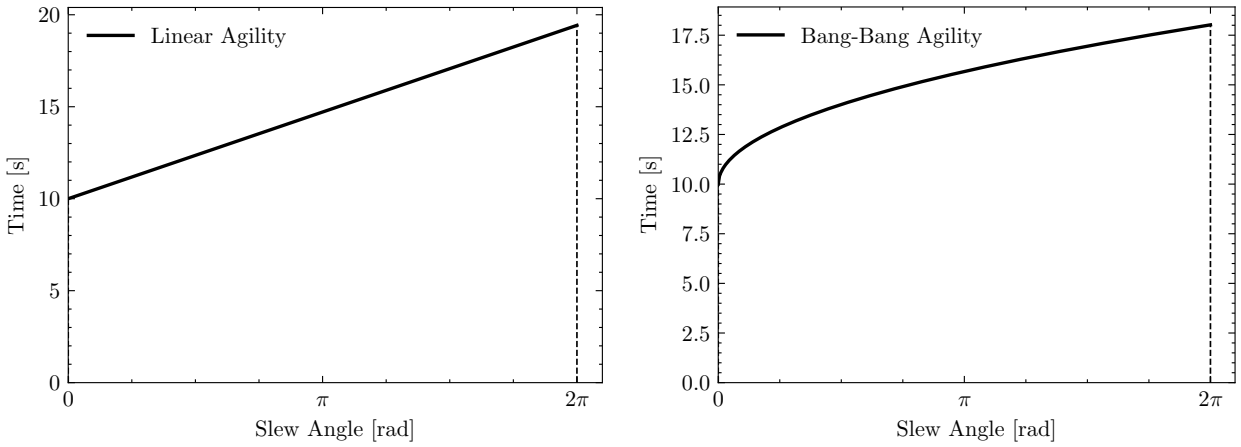
Power on Earth-observing spacecraft is provided by batteries, and replenished by solar panel charging. Power depends on the angle of the sun to the solar panels, operational mode of the spacecraft, which phase of the orbit the spacecraft is in, control of the Maximum Power Point Tracker (MPPT), component temperatures, charging efficiency, and current state of charge of the battery system [119]. Damiani [21] utilizes an energy model which assumes that all energy during daytime operations is covered by the solar panels and that batteries are fully charged going into eclipse. Then power consumption is modeled as permanent and temporary, e.g. bus power is permanent and observations temporarily increase power consumption. Bianchessi [131] does not explicitly model power consumption but instead utilizes a constraint on total duty cycle of the spacecraft that can be dedicated to imaging. Eddy [115] utilizes the expressiveness of a Markov Decision Process (MDP) to formulate power from the solar panels as constant inflow, and observations and slews taking a certain fixed amount of power from the batteries. Herrmann [132] uses the full expressiveness of a tree-search formulation to model power dependent on the incident angle to the sun, with instruments taking a fixed amount of power as a mission parameter.

4.1.2 Data

Data refers to both onboard storage capacity as the source and downlink over ground stations as the sinks. Unlike imaging activities which are distributed over Earth's surface, ground stations are typically placed near the poles due to the high revisit rates [133], leading to smaller windows where downlinks are feasible, approximately 10 minutes long in LEO. Bensena [134] utilizes constraints on onboard storage capacity alone and does not model downlink explicitly. Bianchessi [114] utilizes a greedy downlink modeling approach where images are off-loaded in a first-in first-out fashion with several heuristics examined, including always taking the next downlink opportunity, and taking the next downlink with a probability dependent on the amount of onboard data. Augenstein [133] utilizes a high-fidelity data model which takes into account specific downlink times and data rates along with a required minimum contact frequency with a ground station. As the specific problem formulation used by Augenstein makes simultaneous downlink and image scheduling non-trivial, the optimization program is split by first scheduling downlinks with a heuristic for imaging activity, and then scheduling imaging. A deconfliction algorithm is also used in order to prevent simultaneous downlinks from multiple satellites in a constellation. Eddy [115] utilizes an MDP to model downlinks with a constant data rate with rewards dependent on the length of the downlink; and negative rewards associated with taking downlink actions when not over a ground station.

4.1.3 Agility

Agility is the ability of a spacecraft to re-orient itself. It consists of both the required time to get from one attitude to another, starting at rest and ending at rest, including the settling time required for vibrations in the spacecraft to dampen enough that image quality meets requirements. Optimizing for slewing time is a multi-objective engineering problem, and settling time depends on specific spacecraft structural dynamics, attitude control system [135], and fluid mechanics in the case of spacecraft with a liquid propulsion tank or cooling system. Many researchers abstract agility as a general monotonic function that returns the required time to transition from one task to another [21, 22, 134, 136, 137]. Simplified models include linear agility models such as that used by Eddy [111] and depicted in Figure 4.1a. Linear agility models are certainly not representative of any real-world dynamics of the spacecraft and hence overestimate required slew times in certain regimes, but are extremely cheap to compute and can be useful for extremely large problems consisting of millions of variables and constraints. The next option is to use a constant acceleration model otherwise known as "bang-bang" as depicted in Figure 4.1b which is used by Herrmann [132] with individually modeled reaction wheels. Nag [94, 138] utilizes a bang-bang agility model in conjunction with a proportional-derivative (PD) controller.



(a) Linear agility.

(b) Bang-bang agility (constant torque).

Figure 4.1: Example simplified agility models. Linear agility uses a simple linear model, whereas bang-bang agility uses a constant acceleration model, hence the required time is proportional to the square root of the slew. A constant factor is also added to account for settling time.

The spacecraft must come to rest (within a certain threshold dependent on ground resolution) in order to image a target, meaning that agility cannot be "stored" in the same way that power and data can be. Instead, agility needs to be immediately available to go from one task to another, unlike power and data which can be stored via onboard batteries and storage. If two tasks are too close to each other in time, and there is insufficient agility to transition between them, there is nothing that can be done to schedule that transition. This is markedly different and much more constraining than power and data, where previous actions in the schedule can affect the ability to incorporate new tasks. This can make agility often the tightest constraint

out of all the resources, and it is the main constraint considered in all previous work, as shown in Table 4.1.

4.2 Schedulers and Limitations

Now that constraints and modeling approaches have been defined, we can explore different scheduling algorithms and their respective limitations. Many different methods have been applied to scheduling Earth observation tasks—the problem can be represented in many different ways with different underlying assumptions and limitations. An overview of some of the methods that have been applied in previous research are shown in Table 4.1.

Table 4.1: Review of a selection of constraints and methods considered in spacecraft activity scheduling problems, presented in chronological order. Based on author affiliations, the main ones that are being used to service space operations are Augenstein (2016) [133] and possibly Eddy (2021) [14].

Work	Method	Constraints		
		Agility	Data	Power
Bensana (1999) [134]	MILP	✓	✓	
Wolfe (2000) [136]	Greedy Heuristic + Genetic Algorithm	✓		
Lemaître (2002) [22]	Greedy Heuristic + Graph	✓		
Damiani (2005) [21]	Greedy Heuristic + MILP	✓	✓	✓
Bianchessi (2007) [131]	Greedy Heuristic	✓		
Bianchessi (2008) [114]	Greedy Heuristic	✓	✓	✓
Augenstein (2014) [137]	Graph	✓		
Augenstein (2016) [133]	MILP	✓	✓	
Nag (2018) [138]	MILP	✓		
Nag (2019) [94]	MILP	✓		
Hadj-Salah (2019) [139]	RL (A2C)	✓		
Eddy (2020) [115]	Markov Decision Process	✓	✓	✓
Eddy (2021) [111]	Maximum Independent Set	✓		
Herrmann (2022) [140]	RL (MCTS)	✓	✓	✓
Wang (2023) [141]	RL (Policy Gradient)	✓		✓

4.2.1 Centralized Scheduling

Formulating the spacecraft task scheduling problem as an optimization, the objective can be given simply as

$$O(X) = \sum_{a_i \in \boldsymbol{\sigma}} u_i a_i, \tag{4.1}$$

where $\mathcal{A}$ is the set of imaging opportunities, u_i refers to the utility of an imaging opportunity,

and a_i is a binary variable indicating whether the request is in the schedule, where we wish to maximize the total sum of reward subject to constraints on spacecraft resources.

The simplest method of scheduling in this formulation is from any one step, taking the next best feasible opportunity, where feasibility is derived from resource constraints. This method is referred to as a **greedy algorithm**, or priority dispatch [114, 131, 136] where locally optimal actions are taken at each step based on a heuristic. In temporal space this means taking the next imaging opportunity that is both available and reachable, and has been applied to scheduling the COSMO-SkyMed constellation of synthetic aperture radar (SAR) satellites [114]. Wolfe et al. [136] applied modifications to the greedy algorithm where instead of being greedy directly in the sequence of imaging opportunities, they first mapped the highest-reward activity to be scheduled to its optimal request and then iterated until complete [136]. In the Wolfe approach, the algorithm is greedy in the reward space instead of the action space and performs better than the temporally greedy algorithm, although with longer runtime.

Greedy algorithms are extremely fast to implement due to the fact that they do not backtrack and only do local searches. This is helpful for scheduling problems where compute is limited, such as due to onboard resources. This makes them well-suited for scenarios where rapid decision-making is critical, despite the purely local approach not yielding optimal results.

Mixed Integer Linear Programming (MILP) is an optimization technique that addresses problems that can be represented by linear or integer costs and constraints. Many powerful and well-optimized commercial solvers exist for problems that can be represented as MILPs, which makes them extremely powerful and extensible. MILP-based techniques have wide applicability for machine scheduling problems [142] and many different types of scheduling problems in space [106, 143]. Additionally due to the inherently convex nature of the costs and constraints, MILP solutions can be verifiably optimal, which is often used as a comparison case [111, 137, 139, 144].

For the single spacecraft Earth observation problem, the MILP version of the problem is given by

$$\begin{aligned} \max \quad & \sum_{a_i \in \mathcal{A}} c_i a_i \\ \text{subject to} \quad & a_i, a_j \in \{0, 1\} \\ & a_i + a_j \leqslant 1 \forall k \left(a_i, a_j\right) = 0, a_i, a_j \in \mathcal{A}, k \in \mathcal{K}, \end{aligned} \tag{4.2}$$

where k is an element in the set of constraints $\mathcal{K}$, and x_i, c_i are defined as in Equation 4.1. While the bang-bang agility function itself is non-linear, the program is still linear in the decision variables and hence still a valid MILP. The constraints on agility can be formulated as

$$k_{\text{agility}}\ \left(a_i, a_j\right) = \begin{cases} 1 & \text{if } t_{\text{slew}}(\theta_j - \theta_i) \leq t_j - t_i \\ 0 & \text{otherwise} \end{cases}, \tag{4.3}$$

where $t_{\text{slew}}(\cdot)$ represents the slew time between two roll angles as given by

$$t_{\text{slew}} = t_s + \beta\sqrt{|\Delta\theta|}, \tag{4.4}$$

where β is an agility parameter. θ_j, t_j, θ_i, t_j represent the roll angle and the time of the upcoming task j and the current task i, respectively. Importantly, this formulation abstracts the notion of angular momentum, and implementation of how it is generated or stored, meaning reaction wheels or control moment gyroscopes are equally well modeled.

The repetition constraint can be formulated as

$$k_{\text{repetition}}\ (a_i, a_j) = \begin{cases} 1 & \text{if } r_i = r_j \\ 0 & \text{otherwise} \end{cases}, \tag{4.5}$$

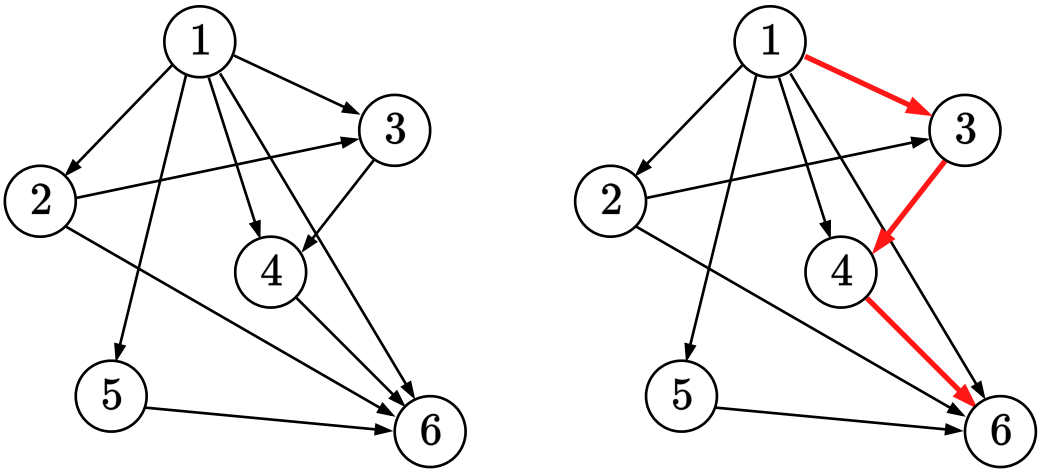
where r_i refers to the individual request associated with an imaging access a_i.

Limitations of MILPs include that all costs and constraints must be linear or integer variables and that the number of decision variables is fixed. Hence, complex agility models beyond the scope of what is discussed above may not be possible at all or require modifications to the problem formulation. Nag [94] utilizes a formulation where the pointing directions of the spacecraft are discretized in order to use a dynamics-based slewing model based on PD control. However this discretization requires many more decision variables to be added to the program for every single imaging opportunity—in the worst case, as many the square of the number of discretized pointing angles—which can impact solve times and requires additional memory [138]. Modifications to this scheduling system have been proposed to run onboard with optimizations such as discretization of orbit propagation into discrete ground points to allow the algorithms to run onboard [94].

Additionally due to the linear nature of the program, scheduling data downlink in proportion to imaging opportunity is not possible without making the program non-linear, as this introduces a dependence on other decision variables. Augenstein [133] mitigates this complexity by splitting up the scheduling problem into two iterative programs: one for imaging and one for downlink, with a heuristic for image activity used in downlink scheduling that links the imaging and downlink programs together.

MILPs can be extended to constellation scheduling by adding sets of binary decision variables for each spacecraft and an additional mutual exclusion constraint for each satellite, ensuring that for each imaging request, only one satellite is used to cover it [94, 133, 138]. Likewise for the single-satellite case, MILPs can find verifiably optimal solutions for constellations, but with more complex programs due to the additional constellation mutual exclusion constraints involved.

Problems relating to decision making can also be represented as **graph traversal**. In the scheduling problem, vertices V can be used to represent imaging opportunities in X and edges E can be drawn between each vertex representing feasible transitions from one vertex to another based on resources. Observations with a continuous decision for start and end times can be discretized to fit into this scheme [137]. Finding the optimal schedule then transforms



(a) Directed acyclic graph (DAG). (b) Longest path through DAG.

Figure 4.2: Example directed acyclic graph (DAG) with 6 nodes, showing (a) transitions and (b) longest unweighted path through DAG.

into finding the longest weighted path in the graph, as this results in the most imaging utility obtained, as depicted in Figure 4.2. Scheduling problems where transitions between vertices V are transitive, i.e., $\forall A, B, C \in V,\ (A \rightarrow B) \wedge (B \rightarrow C) \implies (A \rightarrow C)$ can be represented as a directed graph. Scheduling problems have a natural causality in them as time cannot move backwards, which means that they can be represented as directed acyclic graphs (DAGs), or digraphs.

Due to the transitive requirement on the constraints, only agility constraints can be modeled this way. Since power and data constraints have a memory effect due to batteries and onboard storage, this means that their resources depend on the entire trajectory. If power and data constraints are incorporated into the problem, the graph search problem degenerates from a graph traversal problem to a tree search problem with significantly higher combinatorial complexity.

Graph scheduling methods are not nearly as expressive as MILPs, but are easily interpretable and visualizable, allowing for easy human interaction with the scheduler [137]. Constraints for forcing tasks in and out can be used to manually prioritize certain tasks over others and can be connected to a user interface for rapid manual direction of tasks [137]. Since all activities far into the future are possible in the case of agility-only scheduling, the generated graphs can become extremely dense, with up to 97% edge density in typical cases [14]. A sparsification approach can be used to tackle this combinatorial complexity. Boerkoel et al. [144] used a sparsification approach where any edges longer than a pre-determined planning horizon are not connected, significantly reducing combinatorial complexity.

Graph-based approaches do not naturally extend to constellation planning. Unlike the MILP based solution where more decision variables could be added to the program with mutual exclusion constraints, constellation scheduling with a graph-based scheduling approach requires perioidic re-synchronization of local per-satellite graph searches [14], or doing an iterative improvement after each schedule has been constructed [144]. Performing iterative

solves adds significantly more complexity, which may make alternate methods preferable.

Markov Decision Processes (MDPs) are problems where actions can be taken in a state space based on a transition function, with rewards obtained at each step. MDPs are typically solved by determining the value function of each state and finding a set of state transitions based on the obtained value function, characterizing the long-term value of each state. MDPs are typically discretized at particular timesteps—this approach is followed by Candela et al. [84] for a dynamic targeting problem. For long planning horizon problems, this approach can become computationally intractable. Eddy [115] proposes a

Chapter 5

Lookahead System Design and Drivers

With the pointing requirements and agility specifications defined, this chapter discusses the main system drivers behind lookahead instruments. As mentioned in §3 Mission Concepts, this thesis considers two cases of lookahead instrument: a body-fixed lookahead instrument, and that of using data from meteorological satellites in GEO as a virtual lookahead instrument. For both of these cases, there is a significant amount of spherical geometry and orbital dynamics involved, hence we first introduce the required quantities before analyzing these systems in-depth. Finally, based on the systems analyses as presented, final analysis cases for the lookahead are obtained, which are used in §6 Sensing and Dynamic Tasking for analysis.

5.1 Brief Review of Orbits and Spherical Geometry

Starting from orbital velocity as defined by

$$v_{\text{orbit}} = \sqrt{\frac{\mu}{a}}, \tag{5.1}$$

where μ is the gravitational parameter for Earth, and a is the semi-major axis, which is always equal to $R_E + h$ in the case of circular orbits considered in this thesis. Under these assumptions, the track velocity of the spacecraft projected back onto Earth's surface is

$$v_{\text{track}} = v_{\text{orbit}} \frac{R_E}{a} = \sqrt{\frac{\mu a}{R_E}}. \tag{5.2}$$

Horizon distance and track velocity set a hard limit on how far into the future a lookahead instrument can observe. Straight-line horizon distance shown as line segment SH in Figure 5.1 is given by

$$SH = \sqrt{(R_E + h)^2 - R_E^2}. \tag{5.3}$$

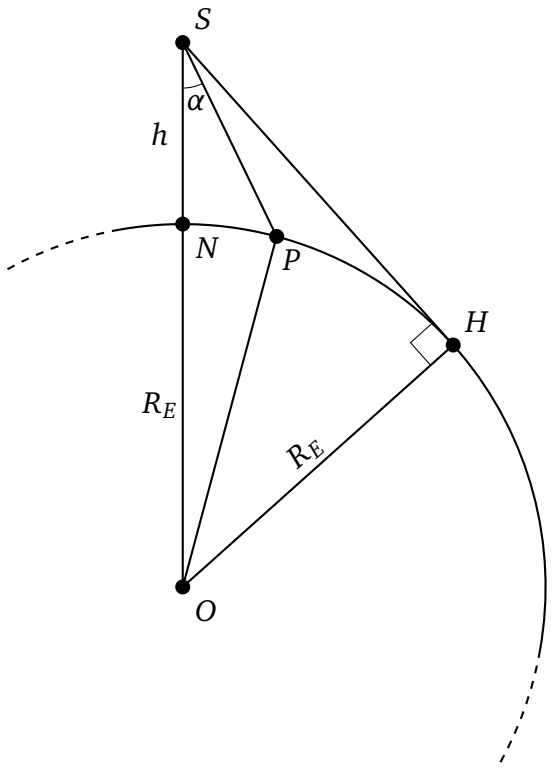


Figure 5.1: Spherical geometry diagram showing Earth, EO spacecraft at S, horizon H, nadir projected point N, and arbitrary point P.

The arc length $\widehat{NH}$ from Figure 5.1 is given by

$$\widehat{NH} = R_E \arccos\left(\frac{R_E}{R_E + h}\right). \tag{5.4}$$

Hence, the total time until a point at the horizon comes to nadir is

$$t_{\text{horizon}} = \frac{R_E}{v_{\text{track}}} \arccos\left(\frac{R_E}{R_E + h}\right), \tag{5.5}$$

Generalizing further, for any arbitrary point P as shown in Figure 5.1, positioned at an angle α pitched in front of the spacecraft at point S, the track distance to nadir N is

$$d_{\text{track}}(\alpha) = R \arccos\left(\frac{(R_E + h)\sin^2\alpha + \cos\alpha\sqrt{R_E^2 - (R+h)^2\sin^2\alpha}}{R_E}\right). \tag{5.6}$$

The total time for any point P situated in front of the spacecraft at S at an angle α to be coincident with the projected nadir point N is

$$t_{\text{point}} = \frac{d_{\text{track}}(\alpha)}{v_{\text{track}}} \tag{5.7}$$

which can be used to size the required vertical field of view of the lookahead subject to timing constraints.

5.2 Vision-Based Instruments

For the case of a body-fixed lookahead sensor, the goal is not only to have sufficient resolution in order to be able to classify pixels in the image, but also with a wide enough field of view to have enough coverage and time to be able to re-optimize the schedule onboard. In this work, we reframe the resolution requirements as a minimization of ground sampling distance, subject to constraints relating to temporal and spatial coverage of the instrument.

We first evaluate timing requirements in terms of timing requirements and translate those into requirements on vertical field of view based on quantities obtained in §5.1 Brief Review of Orbits and Spherical Geometry. Then, we consider constraints on the horizontal field of view based on the predicted vertical field of view.

For the vertical field of view, we make two assumptions:

1. **Spacecraft Neutral Position:** That the spacecraft's neutral position after completing a task while imaging is to point the primary instrument directly towards nadir.
2. **Nadir Edge on Lookahead:** The lookahead instrument's field of view has its bottom edge at nadir, implying that the boresight pitch of the lookahead instrument is always equal to half the vertical field of view.

Figure 5.3 shows how much time it takes for a point at the horizon to come to nadir depending on the angle of lookahead. Some desirable attributes for this lookahead instrument are:

- **Slew Time:** The ability to look ahead of the orbital path with the time it takes to do a full slew between one side of the field of regard to the other, so that all tasks that can be observed are also reachable.
- **Pitching Field of Regard:** The ability to look beyond nadir when in the worst case operational mode, i.e., pitched fully backwards in the nominal field of regard.
- **Settling Time:** Incorporating settling time in the agility models can result in cases where tasks are viewable but no actions can be taken due to the settling time. This implies that there are regions where it is *never* worth performing a lookahead.

For the horizontal field of view, constraints primarily depend on the field of regard. As long as both edges of the field of regard are within the lookahead sensor field of view, and the task utility is unlikely to change significantly between observation in the lookahead and when imaging, then the utility for every task within the field of regard will be observable at least at some time. At minimum, the horizontal field of view must be large enough to view both edges of the field of regard at nadir. Depending on the pitch the angular extent of the field of

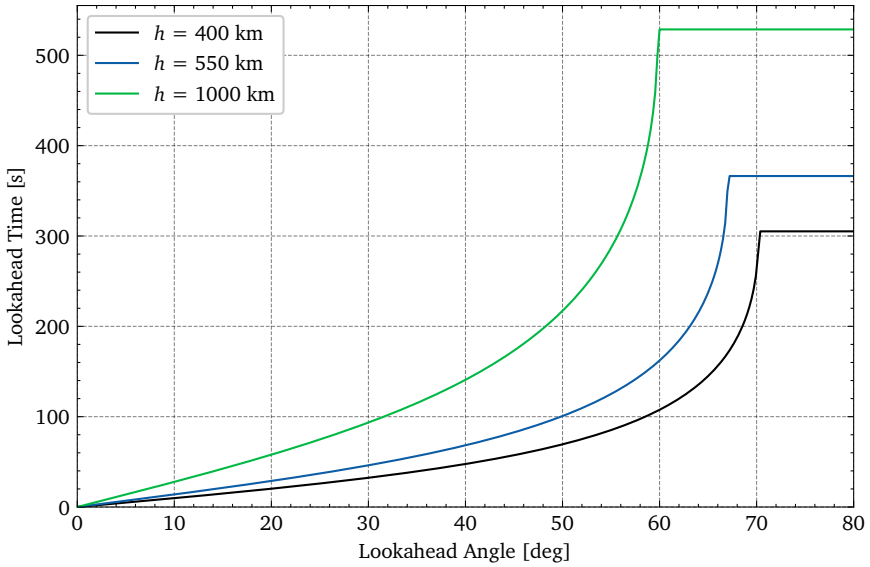


Figure 5.2: Lookahead time depending on angle along-track from nadir for orbital altitudes 400 km (ISS LEO), 550 km (SSO), and 1000 km. Increasing altitude allows for farther lookahead time at a lower angle at the cost of resolution.

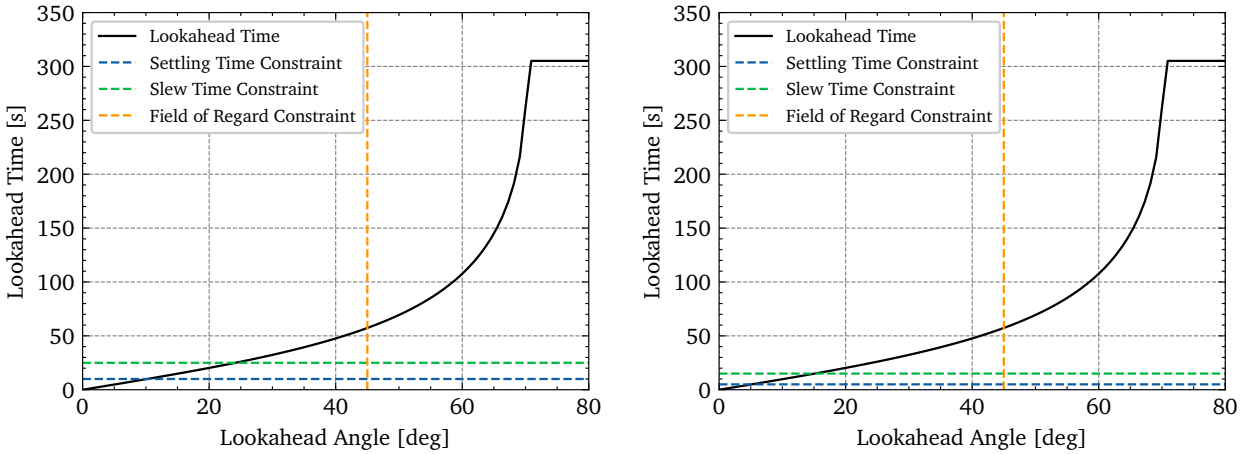
regard can vary. In the extreme case, both edges of the field of regard must also be visible while the spacecraft is rolled at its maximum off-nadir angle for acquiring an imaging access. These horizontal constraints are visualized in Figure 5.4 and summarized as follows:

- **Nadir Field of Regard:** The ability to view both edges of the field of regard while at nadir.
- **Rolled Field of Regard:** The ability to view both edges of the field of regard while rolled over at the maximum off-nadir angle.

The lookahead time for orbital altitudes including 400 km ISS LEO and 550 km SSO are shown in Figure 5.2. Initially, the lookahead time rises almost linearly, before rising much more rapidly around the horizon and abruptly leveling off once the horizon is in view. As orbital altitude increases, the maximum lookahead time along with the angle required to achieve it both decrease, however at the cost of resolution.

The lookahead time along with accompanying vertical field of view constraints are shown in Figure 5.3 for both the agile and ultra-agile cases. For the case of an agile spacecraft with significant off-nadir pointing capability, the tightest constraint is actually on the viewing both edges of the field of regard, which completely overwhelms timing constraints on slewing. The field of regard constraint also necessitates at least approximately one minute of lookahead time. The ground footprint is shown in Figure 5.5 and goes through several regimes, with initially a concave extreme edge which gets exaggerated more and more until the camera is entirely over the horizon, causing that particular edge to become convex, as it snaps to the horizon itself rather than the camera's frame.

Extending these constraints with the horizontal field of view constraints as in Figure 5.6,**??**



(a) Agile.

(b) Ultra agile.

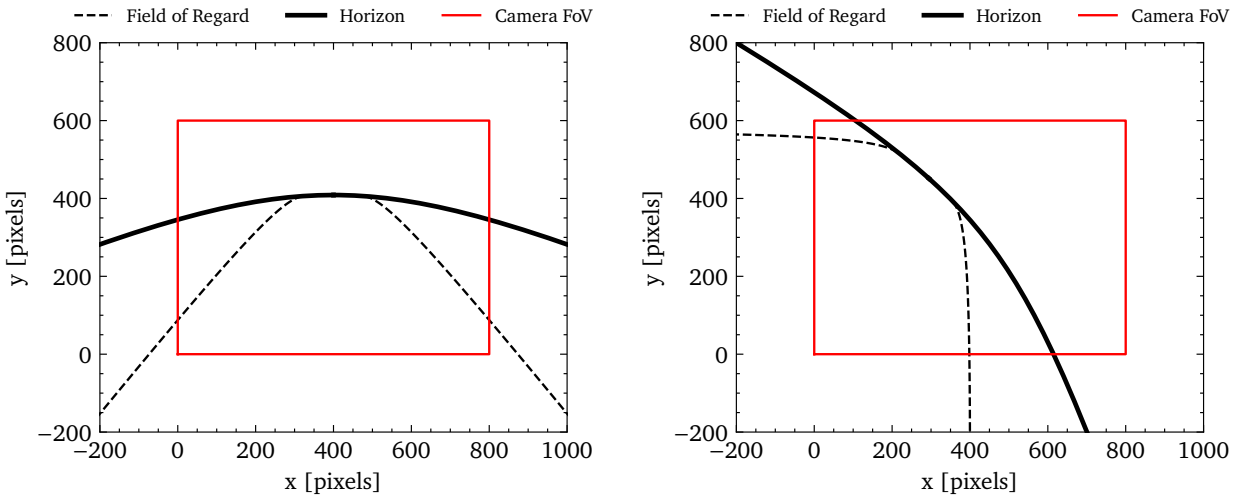
Figure 5.3: Lookahead time constraints for vertical field of view of lookahead instrument for the agile and ultra-agile cases considered in this thesis. Horizontal field of view constraints are the tightest constraints.

the tightest constraints on camera field of view become almost exclusively those based on horizontal field of regard constraints. In the most extreme case where the spacecraft is rolled over to its maximum off-nadir pointing angle, it becomes the only tight constraint.

Table 5.1: Lookahead instrument configurations considered in this work, consisting of a stationary case which does not require additional re-orientation of the spacecraft for full observability of upcoming tasks, and an agile case, which requires re-orientation for full observability.

Case	**Vertical FoV [deg]**	**Horizontal FoV [deg]**	**Boresight Pitch [deg]**
Stationary Lookahead	60	60	30
Agile Lookahead	45	45	22.5

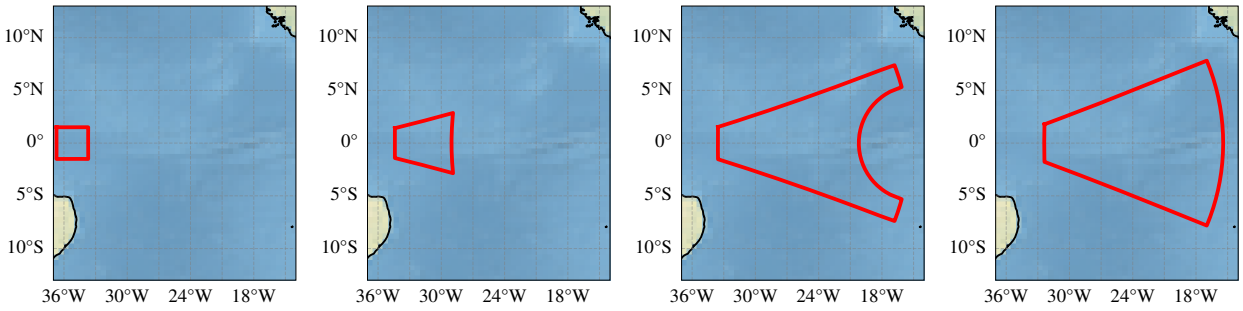
Satisfying all of these imaging constraints allows for dynamic tasking with full observability of upcoming tasks, without requiring any re-orientation of the spacecraft. In this work we consider two cases: one with a wide field of view instrument satisfying all aforementioned constraints that needs no additional re-orientation for fully observable lookahead (stationary case), and a case with a narrower field of view instrument that requires re-orientation of the spacecraft for observability, otherwise referred to in this work as agile lookahead. Lookahead instrument cases are summarized in Table 5.1, with the stationary case using a 60°×60° field of view, and the agile case using a 45°×45°. These cases are also visualized with a simulated camera view from orbit in Figure 5.7, with the same boresight angle, to highlight how much narrower the agile case is.



(a) Set of constraints present at nadir.

(b) Set of constraints present while rolled to maximum extent of field of regard.

Figure 5.4: Highlighted vision constraints on lookahead instrument field of view, (a) at nadir and (b) at extreme roll angles at edge of field of regard. Visualization contains lookahead field of view highlighted in red, with surroundings for context. Objective for lookahead instrument design is to be able to see both field of regard ground track edges at nadir and at maximum roll.



(a) 0° (viewing nadir).

(b) 35° (baseline pitch).

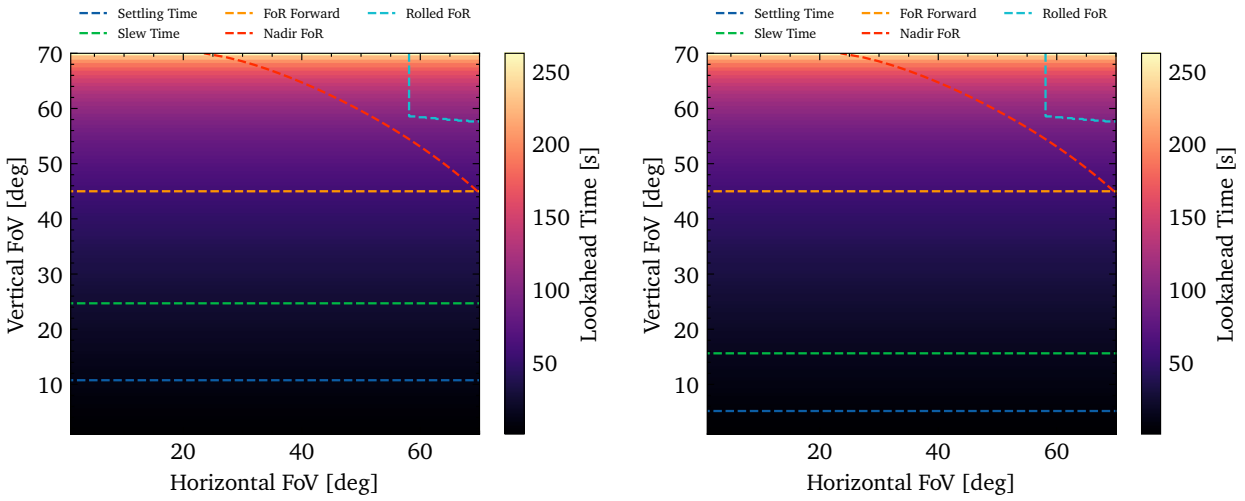
(c) 47° (clipping horizon).

(d) 60° (horizon fully visible).

Figure 5.5: Coverage of lookahead instrument with 45°×45° field of view at various pitch angles.

5.3 Virtual Lookahead with Meteorological Data

Meteorological satellites in GEO provide important observations to inform weather monitoring. Satellites from many different nations cover different regions of the Earth. These satellites are typically equipped with high resolution, high refresh rate scanning instruments with several different wavelength bands. The scanning instruments on these spacecraft can deliver full-disk images up to a rate of every 10 minutes, with specific regions of Earth being scanned even faster than that to allow for tracking of fast-moving weather events [110]. The specific



(a) Agile.

(b) Ultra agile.

Figure 5.6: Lookahead time for a point at an angle away from the spacecraft to come to nadir at 400 km altitude, for ultra-agile and agile spacecraft cases considered in this work. The only difference between these two plots is the set of agility constraints on the bottom.

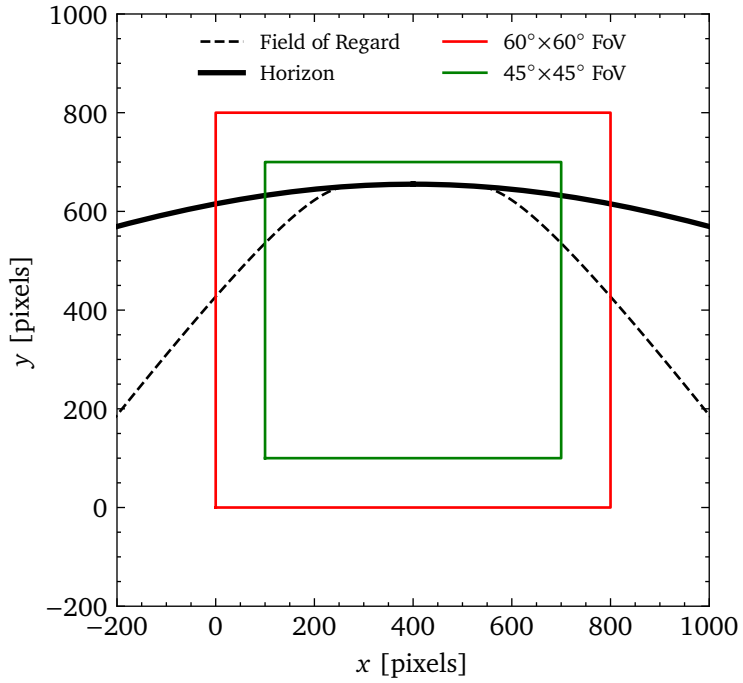


Figure 5.7: Example showing camera projection simulation, along with projected field of regard, horizon, and camera field of view for 60°×60° stationary case and 45°×45° agile case. Both views are shown at the same boresight pitch of 50°.

service regions, satellites, instruments, center longitudes, and full-disk image cadences used in this work are provided in Table 5.2. The associated instruments, their available bands, and

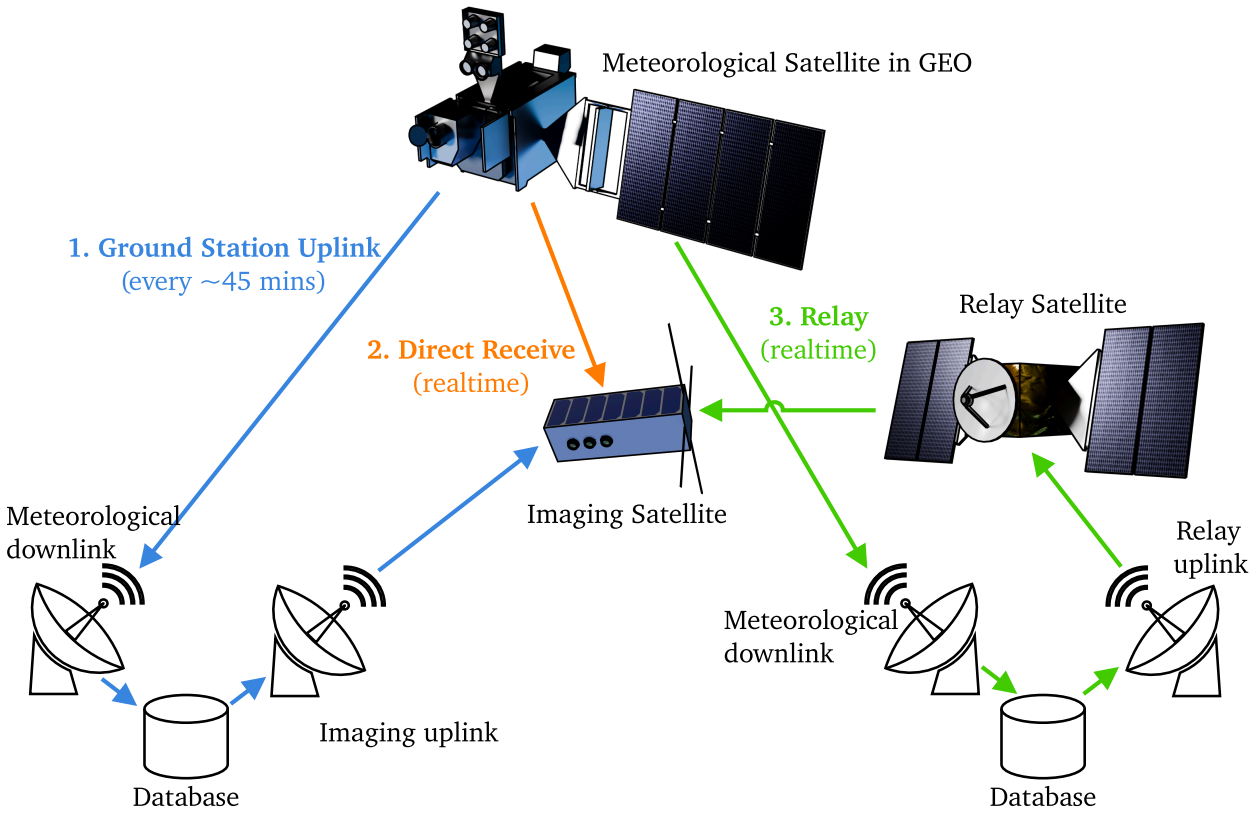


Figure 5.8: Diagram showing methods by which data from meteorological satellites can be transferred to imaging satellites. (1) Data can be relayed through downlinking, extracting from a database, and uplinking to the satellite, resulting in communication gaps. (2) Data can also be obtained in near realtime through direct reception by the imaging satellite through an antenna, or (3) relayed through a network such as Iridium, GlobalStar, or the NASA Tracking and Data Relay Satellite (TDRS) system.

nadir GSD are provided in Table 5.3.

For the case of a virtual lookahead sensor using meteorological from GEO data, there is significantly less systems analysis on the instrument side, as all the instruments are already flying onboard current meteorological spacecraft. However, while the instrument configuration is fixed, locating the lookahead instrument away from the spacecraft introduces additional complexity in terms of data transfer between the meteorological spacecraft and imaging spacecraft.

There are three main ways that data can be processed and transferred from a meteorological satellite to an imaging satellite, as shown in Figure 5.8. Images from the meteorological satellites are downlinked through a high-rate radio, that allows for L1 imagery to be available through storage systems such as through Amazon S3 within minutes after capture and downlink [110], with processed L2 imagery being processed and delivered a few hours later. Since the data is publicly available, it can be retrieved by the imaging satellite's ground station, and then either (1) uplinked via the ground station, or (3) relayed by another satellite through

Table 5.2: Selected meteorological satellites in GEO providing full-disk imagery, with associated instrument, center longitude, and full-disk image cadence, current as of 2025-04-01. GOES-19 replaced GOES-16 as the operational GOES East service spacecraft on 2025-04-07.

Service	Satellite	Instrument	Center Longitude	Full-Disk Cadence
GOES East [145]	GOES-16	ABI	75.2° W	10 minutes
GOES West [145]	GOES-18	ABI	137.2° W	10 minutes
Meteosat ZDS [146]	Meteosat-10	SEVIRI	0° E	15 minutes
MeteoSat IODC [146]	Meteosat-9	SEVIRI	45.5° E	15 minutes
Himawari [147]	Himawari-9	AHI	140.7° E	10 minutes

Table 5.3: Table of instruments, available bands, and nadir GSD range from instruments onboard satellites listed in Table 5.2.

Instrument	Bands [µm]	Nadir GSD [km]
ABI (GOES) [148]	0.47, 0.64, 0.86, 1.37, 1.6, 2.2, 3.9, 6.2, 6.9, 7.3, 8.4, 9.6, 10.3, 11.2, 12.3, 13.3	0.5 – 2.0
SEVIRI (Meteosat) [149]	0.4 – 1.1 broad, 0.60, 0.81, 1.60, 3.90, 6.25, 7.35, 8.70, 9.66, 10.8, 12.0, 13.4	1 – 3.0
AHI (Himawari) [147]	0.47, 0.51, 0.64, 0.86, 1.60, 2.3, 3.9, 6.2, 6.9, 7.3, 8.6, 9.6, 10.4, 11.2, 12.4, 13.3	0.5 – 2.0

a system like Iridium, GlobalStar, or the NASA Tracking and Data Relay Satellite (TDRS) system. Additionally, through low-rate channels such as through GOES' High Rate Information Transmission / Emergency Managers Weather Information Network (HRIT/EMWIN) low-rate broadcast system, multispectral imagery can also be (2) directly downlinked to the imaging satellite, although this requires additional licensing and hardware [109]. Using either (2) or the (3) allows real-time decision making, whereas communication gaps from a ground station means updates can only be transferred approximately every 45 minutes for an imaging satellite in LEO.

We consider data from the Geostationary Operational Environmental Satellites (GOES) program, the Meteosat Second Generation (MSG) program, and the Himawari program. Figure 5.9 shows the total theoretical coverage of these spacecraft along with the largest 10,000 world cities dataset used in this work [116]. Figure 5.9 shows that the theoretical coverage of these spacecraft spans the globe, up to approximately ±80 degrees latitude, although with ground sample distance (GSD) significantly decreasing at grazing incidences, starting at 1 km for most bands at nadir and 10+ km at imaging limits, as visualized in Figure 5.11. Figure 5.10 shows the same coverage, but grouped by minimum imaging cadence, where the majority of the Earth is served by satellites with 10-minute coverage, with areas of Europe, Africa, and the Middle East only affording 15-minute coverage. L2 data such as binarized cloud masks however are typically more limited in coverage as compared to L1 data, due to the complexity of obtaining reflectance values for grazing incidences. Cloud masking algorithms from these spacecraft can rely on emissive bands as well as reflective bands, hence cloud masks can be obtained even at night, allowing for global coverage, although with artifacts along the day/night terminator line due to rapidly changing irradiance values from the sun.

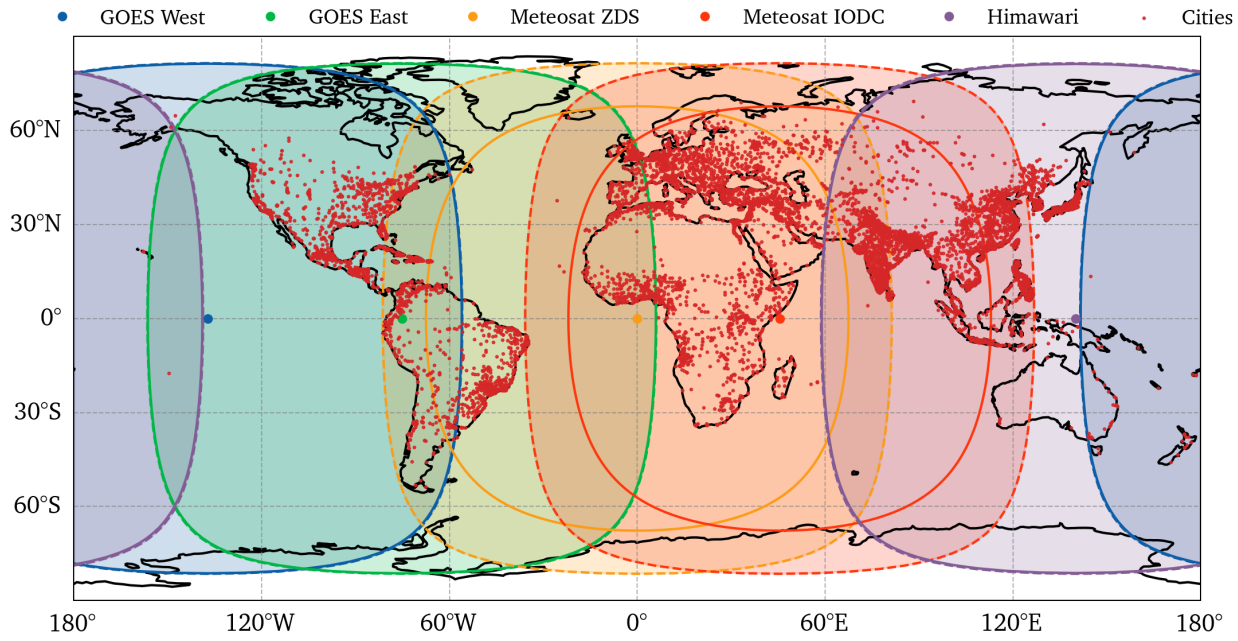


Figure 5.9: Theoretical (dotted) and actual (solid) coverage of selected meteorological satellites. Actual data coverage of products such as cloud masks is typically narrower, due to complexities of deriving reflectance data at grazing incidences. The largest 10,000 world cities used in this work as an imaging target dataset are also overlaid [116].

5.4 Summary

This chapter conducted systems analysis for the design of lookahead instruments for dynamic tasking. We introduced the orbital and geometric relationships that bound the amount of lookahead time available and defined constraints that govern field of view requirements. From these, two representative instrument configurations were defined: a 60°×60° stationary case that achieves full observability without re-orienting the spacecraft, and a narrower 45°×45° case that requires spacecraft maneuvers, referred to as agile lookahead.

We also considered the use of meteorological satellites in GEO as virtual lookahead instruments, highlighting their cadence, coverage, and data-transfer pathways. While such external sensors reduce spacecraft payload complexity, they introduce latency and communication challenges. Together, these analyses define the baseline lookahead cases used in subsequent dynamic tasking evaluations.

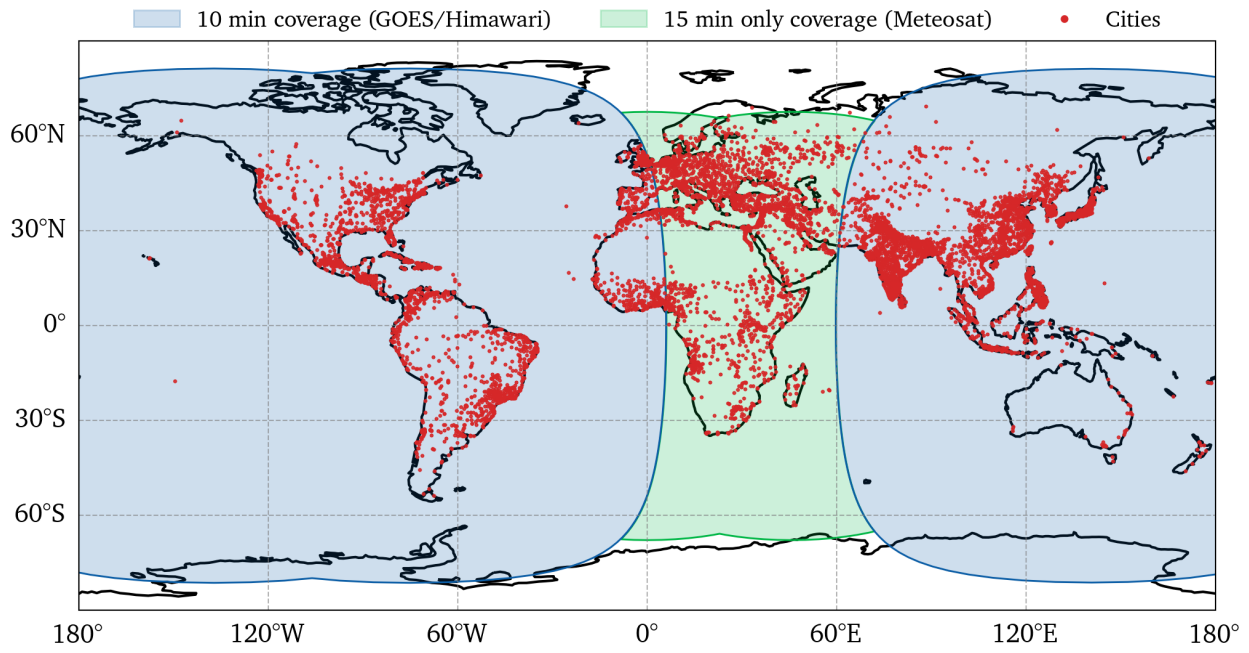


Figure 5.10: Global GEO satellite coverage map organized by the most frequent full-disk image cadence from meteorological, with 10,000 world cities dataset overlaid. The majority of Earth's surface is covered by satellites that can give 10-minute full-disk image cadence, except for parts of Europe, Africa, and the Middle East, which are exclusively covered by Meteosat and can do only a 15-minute full-disk image cadence.

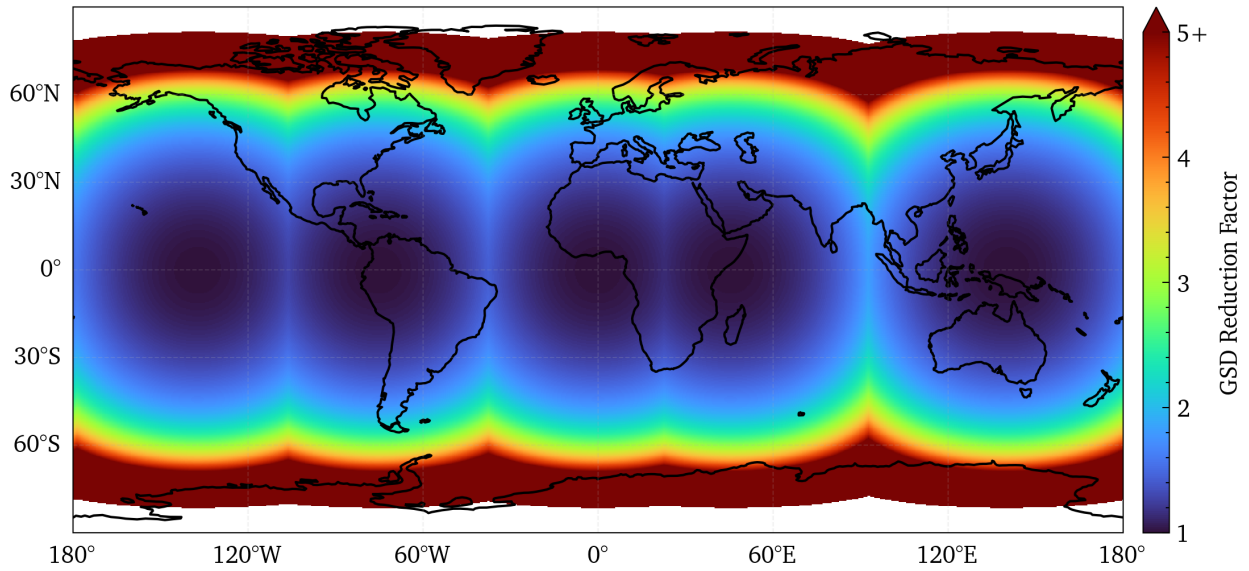


Figure 5.11: World map of minimum GSD reduction due to grazing incidences based on satellites in Table 5.2. Reduction is caused by grazing incidences smearing pixels over larger ground surface. GSD reduction is minimized at nadir, obtaining true GSD values.

Chapter 6

Sensing and Dynamic Tasking

In this chapter, we take all the concepts introduced in previous chapters and piece them together in a dynamic tasking formulation. In this chapter, the spacecraft and instrumentation cases used in this work are summarized, the exact problem definition for dynamic tasking is introduced, a method by which existing scheduling algorithms for spacecraft are extended to enable dynamic tasking, camera simulation and software details are given, methods for onboard schedule repair are introduced, and results with real-world task information and states are aggregated. It is important to note that the analysis presented here, while extensible, is a point analysis on a highly parameterized space—there are a large number of mission concepts and drivers not considered in this work as the space is highly parameterized.

6.1 Dynamic Tasking Problem

In this work, we consider dynamic tasking distinct from dynamic targeting, where dynamic tasking maintains an initially scheduled solution. We can then consider dynamic tasking a different, but related problem to the scheduling problem denoted in §4 Satellite Resources, Constraints, and Scheduling. Scheduling itself can be considered a form of orienteering problem, where a subset of nodes in a graph must be visited to maximize a set of known rewards, similar in concept to a Knapsack problem combined with a Traveling Salesman Problem (TSP) [150]. Dynamic tasking replaces known rewards with stochastic rewards and also adds the aspect of "observing" upcoming tasks within a horizon. This modified problem can be considered an Adaptive Stochastic Orienteering Problem (ASOP) [151, 152]. The full-fidelity ASOP can be described as a Partially Observable Markov Decision Process (POMDP). The state s_t is defined as

$$s_t = \{\boldsymbol{\Theta}, \mathbf{r}, \boldsymbol{\sigma}\}, \tag{6.1}$$

where $\boldsymbol{\Theta}$ represents the current roll, pitch, and yaw of the spacecraft, and $\mathbf{p}$ is a vector of tuples, denoting all of the spacecraft accesses and its belief state (i.e. clear or cloudy), and $\boldsymbol{\sigma}$ represents the current schedule. The set of actions, or decisions, at any time t, d_t, is defined as

$$d_t \in \{\text{slew}, \text{observe}, \text{image}\}. \tag{6.2}$$

This exact form of the problem is provided primarily as a formal reference and will not be used in this work, as the combinatoric space balloons massively and becomes computationally intractable with a large number of tasks. In this work we instead first use the schedule obtained from a conventional MILP solver, and then interleave lookahead maneuvers greedily, based on where a heuristic shows that the schedule is likely to improve. The MILP can be used also as a benchmark, as if we give it information about access states, it will produce an optimal schedule as if it has perfect futuresight, representing a theoretical upper bound on performance.

6.2 Approach

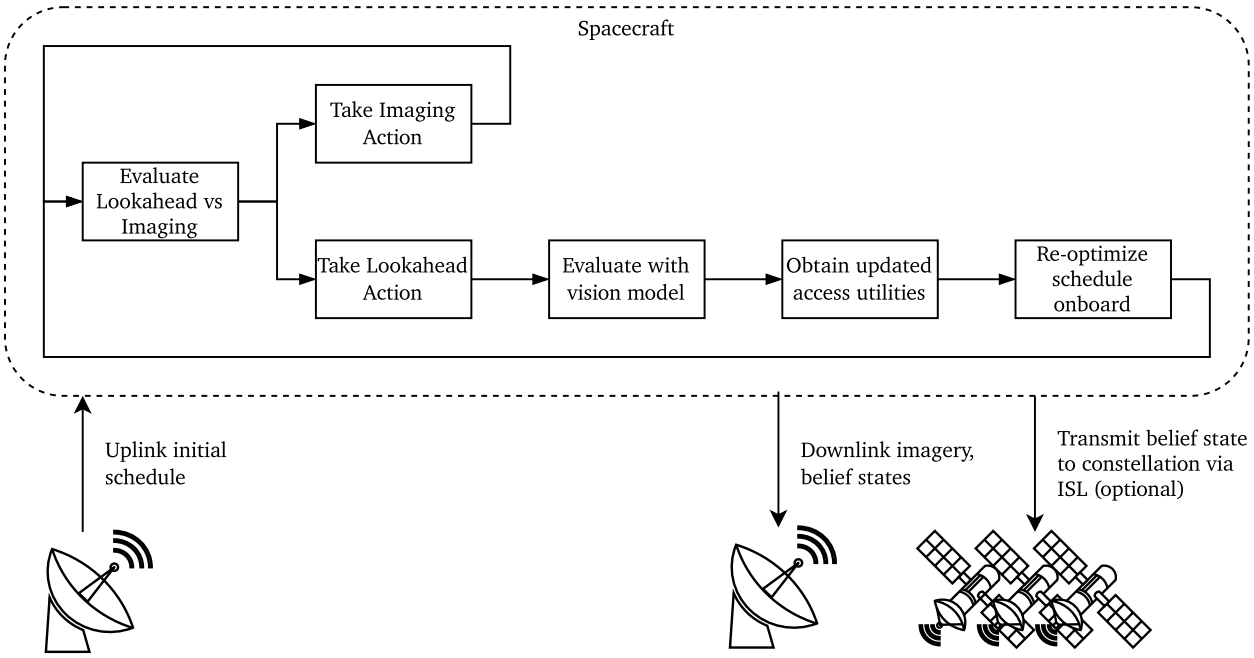


Figure 6.1: Dynamic tasking with body-fixed lookahead framework with initial acquisition, analysis, and re-optimization of schedule onboard spacecraft.

There are many potential approaches and methods for dynamic tasking algorithms, ranging from extensions to existing scheduling methods as described in §4 Satellite Resources, Constraints, and Scheduling, up to entirely end-to-end learned methods. This work focuses on dynamic tasking algorithms that are operationalizable–hence, due to the often stochastic nature of learned methods and the existing infrastructure in place for spacecraft scheduling, and the desire for verifiable and deterministic algorithms running on spacecraft, dynamic tasking is approached as an extension of existing scheduling methods.

Linking these two approaches for spacecraft scheduling itself is nontrivial. While spacecraft scheduling methods themselves can be described as an orienteering problem and solved as

a MILP efficiently, the lookahead actions essentially involve tying in an *additional* NP-hard set-covering problem into the mix. The additional set covering actions add statefulness that balloons combinatoric space and requires consideration of long trajectories when tying into the scheduling problem, as well as continuous states that can be difficult to analyze. With these massive computational requirements for the combined problem, deployment and real-time requirements become extremely difficult to meet, even on the most modern edge compute hardware in space.

Instead, we approach dynamic tasking as two separate problems that are linked together by a heuristic: we extend previous work on scheduling, with a lightweight heuristic that can be run onboard the spacecraft that can decide at key intervals whether to continue with the existing schedule, or to take a lookahead action, if the expected benefit of the lookahead action is positive. After taking a lookahead action, an onboard scheduler can then re-optimize imaging activities based on updated utilities that may come from a vision model, such as that of a convolutional neural network (CNN) or similar.

After updating access utilities, the onboard schedule can then be re-optimized. While the simplest case involves simply removing imaging activities from the schedule that are marked as cloudy, in this work we model the spacecraft as also having an onboard priority queue of alternate image accesses, perhaps one that did not make it into the main schedule optimized on the ground, that can be used as alternatives and slotted into the schedule based on overall utility. Figure 6.2 shows the data model used onboard the spacecraft in this work, containing both the pre-optimized schedule and a priority queue of requests, and Figure 6.3 shows the approach used in this work, where the schedule is re-optimized locally using a MILP scheduler, and progressively gets re-optimized based on a receding horizon as the spacecraft continues through time.

Once the spacecraft has processed data from lookahead actions and obtained updated task utilities, and updated the schedule, in this work we assume that there are two main ways to share the updated belief state and schedule with the rest of the satellite constellation: either through an inter-satellite link (ISL) that is always available, or through transmitting the data over a ground station and having other spacecraft receive the data on uplink.

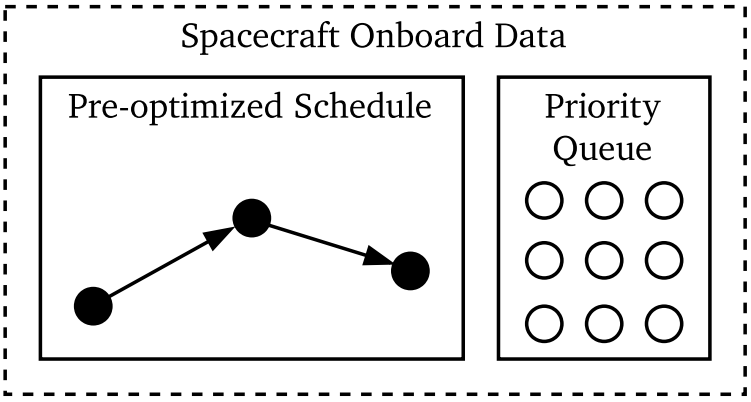


Figure 6.2: Spacecraft onboard data consisting of pre-optimized schedule obtained from the ground scheduler and a priority queue of tasks that can be slotted in.

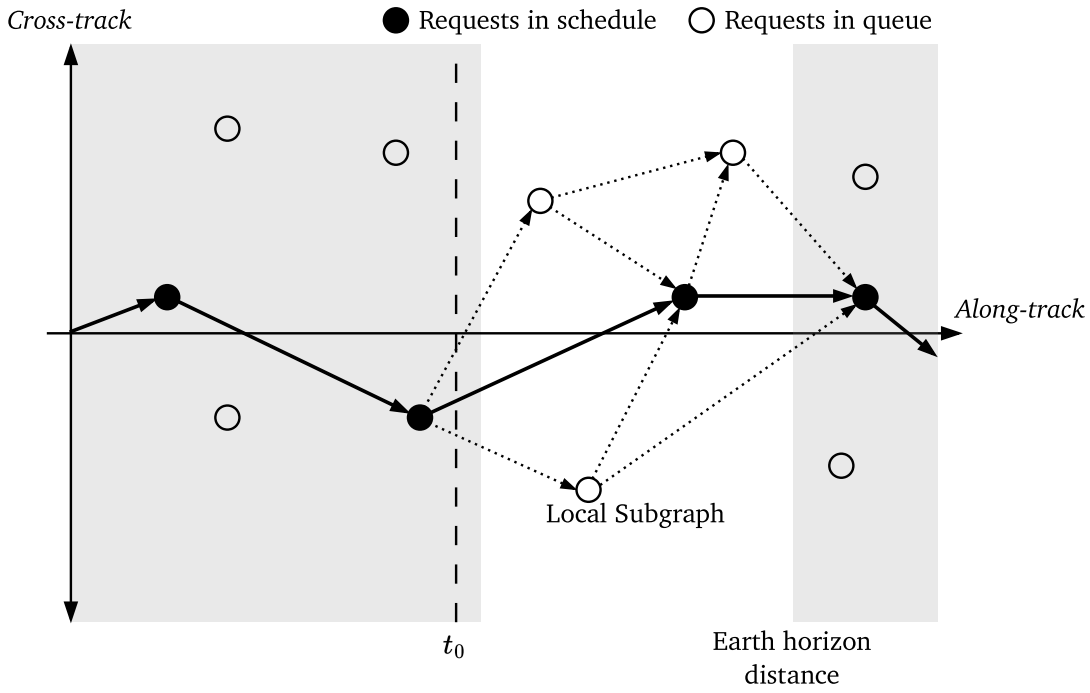


Figure 6.3: Diagram of proposed subgraph to be optimized, which lies between the spacecraft's frozen schedule period and Earth's horizon i.e., the region in which lookaheads can provide new information.

6.2.1 Orbit Propagation

The orbit propagator is one of the most fundamental components and every subsequent component is reliant on it for co-ordinate references and transformations. Since orbits considered in this work are purely circular, with short time horizons of maximum duration 6 hours, a simple Keplerian orbital propagator has been used, with additional flexibility to incorporate a more robust propagator such as SGP4 in the software.

6.2.2 Access Aggregation

The access aggregation algorithm is adapted from [14], specialized specifically for the point accesses and circular orbits considered in this work. The algorithm takes in an input of imaging request latitude and longitude coordinates along with orbital elements, and returns a set of accesses which are a tuple of the request itself, time, and the roll angle of the spacecraft required to image that particular access, with the implicit assumption that all imaging accesses are taken at a pure roll angle, with zero pitch involvement.

The algorithm runs in two steps: a coarse step to find potential accesses, and then a fine step, which uses a recursive binary search over timestep to obtain the exact access time within a specified tolerance, and then a final step to obtain the access roll angle once the time is known.

The coarse algorithm is shown in Algorithm 1. For each timestep all requests are transformed into ECI co-ordinates at that timestep and one timestep in the future. Then, these transformed requests are culled, based on a preset filter radius. The radius varies by timestep size, sensor field of view, and more, as it must capture the worst case in which an access can move in between the two timesteps, sensor field of view, and Earth's rotation. The filter radius is given by

$$\rho_{\text{filter}} = \sqrt{\left(\tfrac{1}{2} v \Delta t\right)^2 + \left(h + R_E \left(1 - \cos \theta_{\text{tot}}\right)\right)^2 + \left(R_E \sin \theta_{\text{tot}}\right)^2}, \tag{6.3}$$

which is an L2 norm of all distances involved, where the worst-case geocentric motion θ_{tot} is given as

$$\theta_{\text{tot}} = \arctan\left(\frac{h \tan \theta_{\text{FoR}}}{R_E}\right) + \gamma \Delta t, \tag{6.4}$$

which involves an approximation of the geocentric angle subtended by the field of regard limits, and Earth's rotation rate in the sidereal frame γ, with the value of 360.985 61 °/day used in this work [119]. While the geocentric angle approximation is in the culling filter somewhat of an under-approximation, in practice given that the along-track distance is an over approximation, the algorithm is still consistent even in extreme cases of polar orbits. After filtering, the signed distances between the planes coincident with the ECI coordinates r and normal to the velocity vector v in ECI coordinates for each time bounding the timestep. Each access that satisfies these criteria is then sent to a recursive binary search algorithm with the planes and distances as initial inputs.

The second step of the algorithm, the recursive binary search, is shown in Algorithm 2. For an access to be purely in roll, the aforementioned plane must contain the access. Hence, the signed plane distances are used with a binary search to minimize the timestep that defines these two planes, recursing until either no accesses are within the planes, or the timestep approaches a specified tolerance. We also only consider accesses under illumination, not in eclipse. This results in separation between blocks of activities, which improves dynamic tasking performance, as there is essentially areas of time where the only useful action is to perform a lookahead.

This approach is trivial to parallelize, as a worker or separate thread can be initialized for each separate coarse chunk or recursive search. This technique is also trivial to extend to constellations, as it can be run once per available orbit, returning a per-spacecraft list of accesses, which can then be fed into a MILP scheduler as described in §4 Satellite Resources, Constraints, and Scheduling, treating the repetition constraint still as per request, which generalizes across the constellation.

Algorithm 1: Access-window detection via coarse pre-filtering and recursive plane-crossing search

Input:
$\mathcal{R}$: set of request lat/long pairs ;
O: orbital elements ;
θ_{FoR}: instrument field of regard ;
Δt: coarse time step ;
t_s, t_e: analysis start and end epochs
Output: $\mathcal{A}$: set of feasible accesses (j, t, α)

$\mathcal{A} \leftarrow \emptyset$; `// collect all accesses`
Pre-compute filter radius $\rho\,(\theta_{\text{FoR}}, \Delta t, O)$;
for $k \leftarrow 0$ **to** $\lfloor (t_e - t_s)/\Delta t \rfloor - 1$ **do** `// scan timeline in coarse chunks`
  $t_1 \leftarrow t_s + k\,\Delta t$; $t_2 \leftarrow t_1 + \Delta t$;
  Propagate orbit to $\mathbf{r}_1, \mathbf{v}_1$ at t_1 and to $\mathbf{r}_2, \mathbf{v}_2$ at t_2;
  `// coarse geometric culling`
  foreach $j \in \mathcal{R}$ **do**
    $\mathbf{p}_{j,1} \leftarrow$ LatLong2ECI(j, t_1);
    $\mathbf{p}_{j,2} \leftarrow$ LatLong2ECI(j, t_2);
    if $\min\left(\|\mathbf{p}_{j,1} - \mathbf{r}_1\|, \|\mathbf{p}_{j,2} - \mathbf{r}_2\|\right) > \rho$ **then** `// discard clearly unreachable requests`
      remove j from further consideration;
  `// detect plane crossings`
  foreach *remaining target* j **do**
    $d_{j,1} \leftarrow$ SignedPlaneDist $\left(\mathbf{r}_1, \mathbf{v}_1, \mathbf{p}_{j,1}\right)$;
    $d_{j,2} \leftarrow$ SignedPlaneDist $\left(\mathbf{r}_2, \mathbf{v}_2, \mathbf{p}_{j,2}\right)$;
  $C \leftarrow \{\, j \mid d_{j,1} \cdot d_{j,2} < 0 \ \wedge\ d_{j,1} > d_{j,2} \,\}$ `// crosses plane inward`
  `// refine with recursive binary search`
  foreach $j \in C$ **do**
    $\mathcal{A} \leftarrow \mathcal{A} \cup$ RecursiveSearch$(j, t_1, t_2, \mathbf{r}_1, \mathbf{r}_2, d_{j,1}, d_{j,2})$
return $\mathcal{A}$

Algorithm 2: RecursiveSearch$(j, t_1, t_2, \mathbf{r}_1, \mathbf{r}_2, d_1, d_2)$

Input:
target j and bounding information from Algorithm 1 ;
t_{tol}: temporal termination tolerance ;
Output: zero or more accesses for j in (t_1, t_2)
if $t_2 - t_1 < t_{tol}$ **then** `// temporal resolution reached`
  $t^\star \leftarrow (t_1 + t_2)/2$; propagate to $\mathbf{r}^\star$; rotate target $\mathbf{p}^\star$; $\mathbf{r}^\star \leftarrow$;
  PropagateOrbit$(O, t^\star)$;
  $\mathbf{p}^\star \leftarrow$ LatLong2ECI$(j, t^\star)$;
  $\alpha \leftarrow \measuredangle(\mathbf{r}^\star, \mathbf{p}^\star - \mathbf{r}^\star)$;
  if $|\alpha| < \theta_{\text{FoR}}$ ***and*** *target not in eclipse at* $t^\star$ **then**
    return $\{(j, t^\star, \alpha)\}$
  return $\emptyset$
$t_3 \leftarrow (t_1 + t_2)/2$; propagate to $\mathbf{r}_3, \mathbf{v}_3$; rotate target to $\mathbf{p}_{j,3}$; compute d_3 ;
return *RecursiveSearch*$(j, t_1, t_3, \mathbf{r}_1, \mathbf{r}_3, d_1, d_3) \cup$ *RecursiveSearch*$(j, t_3, t_2, \mathbf{r}_3, \mathbf{r}_2, d_3, d_2)$

6.2.3 Initial Schedule

We use the same techniques for generating initial schedules as described in §4 Satellite Resources, Constraints, and Scheduling, using a MILP scheduler with constraints on repetition, and agility cases as in Table 3.3.

6.2.4 Access Utility

We use meteorological data of stitched and binarized cloud masks as shown in Figure 6.4 to assess true access states and hence their utility. Intrinsic utility is either constrained to be unit or Pareto-distributed and modified by the cloud state, which is nearest neighbor sampled on the cloud mask. While meteorological data has limits on latitude imposed by viewing angle and orthorectification constraints, of the 10,000 world cities used as a dataset in this work, only three are outside coverage of the meteorological satellites considered, and are excluded for further analysis, as a true utility cannot be obtained. The data coverage zones are shown in Figure 5.10.

6.2.5 Camera Projection and Vision System

We use conventional camera projection matrices for simulation, with projections in homogeneous co-ordinates given by [153]

$$\mathbf{x} = \mathbf{KPX}, \tag{6.5}$$

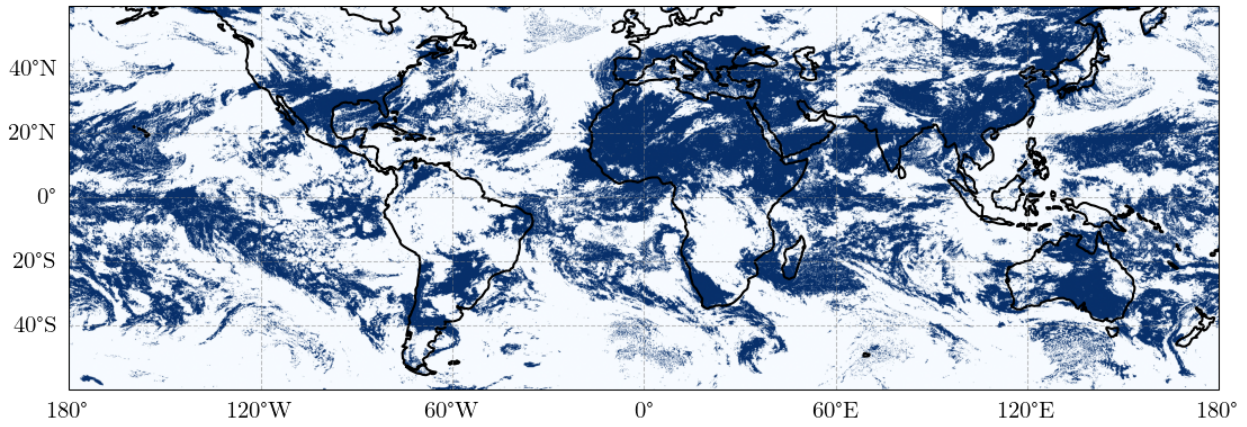


Figure 6.4: Example binary cloud mask from stitching observations from selected meteorological spacecraft at time 2025-01-01T00:00:00+00. White areas denote cloudy regions, blue areas denote non-cloudy regions, and coastlines are highlighted in black.

where $\mathbf{K}$ is the camera intrinsics matrix given by

$$\mathbf{K} = \begin{bmatrix} -f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.6}$$

where f_x, f_y are the focal length of the imaging system, and c_x, c_y are the sensor co-ordinate offsets, which are half the width and height in pixels, respectively. The extrinsics matrix $\mathbf{P}$ is given by

$$\mathbf{P} = \begin{bmatrix} \mathbf{R} & -\mathbf{RT} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{6.7}$$

where $\mathbf{R}$ represents the rotation matrix between the spacecraft's body frame and the ECI frame, and $\mathbf{T}$ represents the translation between the spacecraft's body frame and the ECI frame. The convention shown above produces image projections with the origin at the bottom left corner, of which an example is shown in Figure 6.5.

6.2.6 Lookaheads and Schedule Repair

While the onboard scheduling system need not be identical to the ground scheduling system, they are still innately connected since they are optimizing around the same resources. In this section I will discuss onboard schedule re-optimization in the context of resource and compute limitations.

For onboard execution of dynamic tasking, it is assumed that along with the pre-optimized schedule, there is a priority queue of accesses that could not make it into the initial schedule that the onboard tasking graph is constructed from as depicted in Figure 6.2. The tasks initially

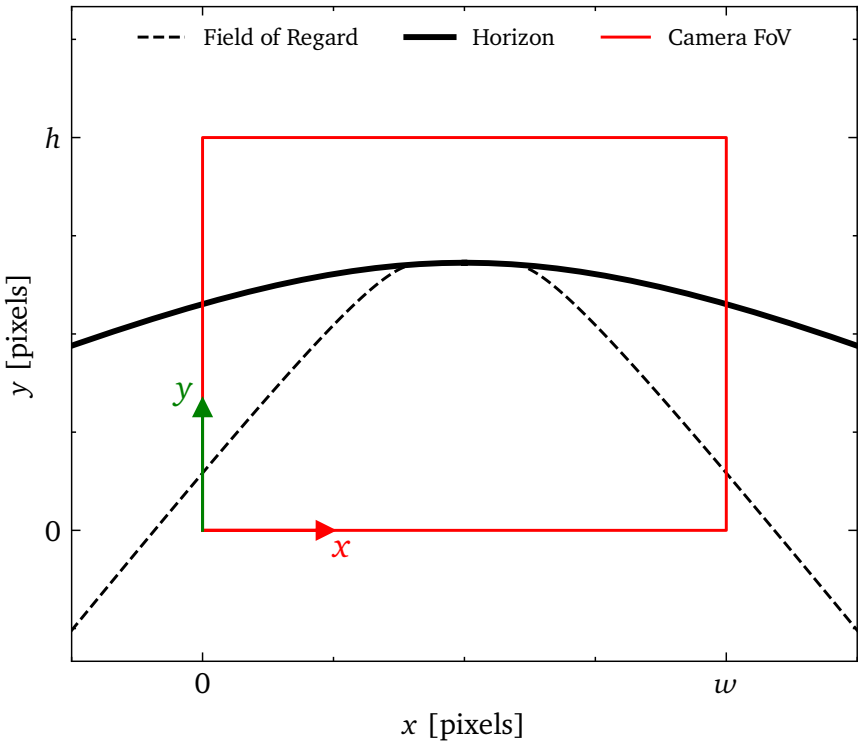


Figure 6.5: Example showing camera projection axes conventions, along with projected field of regard, horizon, and camera field of view .

will be selected as all the leftover imaging requests, but for application onboard a commercial spacecraft there is likely a ranking that can be used so that excessive data is not uploaded to the spacecraft.

Utilizing this data approach and combining it with receding horizon planning techniques as proposed by Boerkoel et al. [144], allows for optimization over only a local subgraph that connects back to the original pre-optimized schedule, as shown in Figure 6.3. In this case the natural planning horizon is the distance through which the lookahead instrument can actually update utility values, i.e., the actual Earth horizon distance of the spacecraft.

The onboard scheduling system utilizes a technique similar to [94] and [133] in utilizing a heuristic to link the scheduling system of different actions together. Utilizing the lookahead heuristic as described in , the lookahead action can then be described as any other activity and be incorporated into the schedule. This method of using the heuristic allows for completely abstracting the complication of observation being coupled with spacecraft state and then allowing any scheduling method to schedule both lookahead and imaging activities, as depicted in Figure 6.7. In theory if the heuristic gives an estimated action utility a positive value, that action should be taken. However, the heuristic can only consider local actions. The simplest way to incorporate some level of global planning and awareness is to not took a lookahead action below a specific threshold.

While it may be preferable to have the lookahead actions preferentially target tasks that are

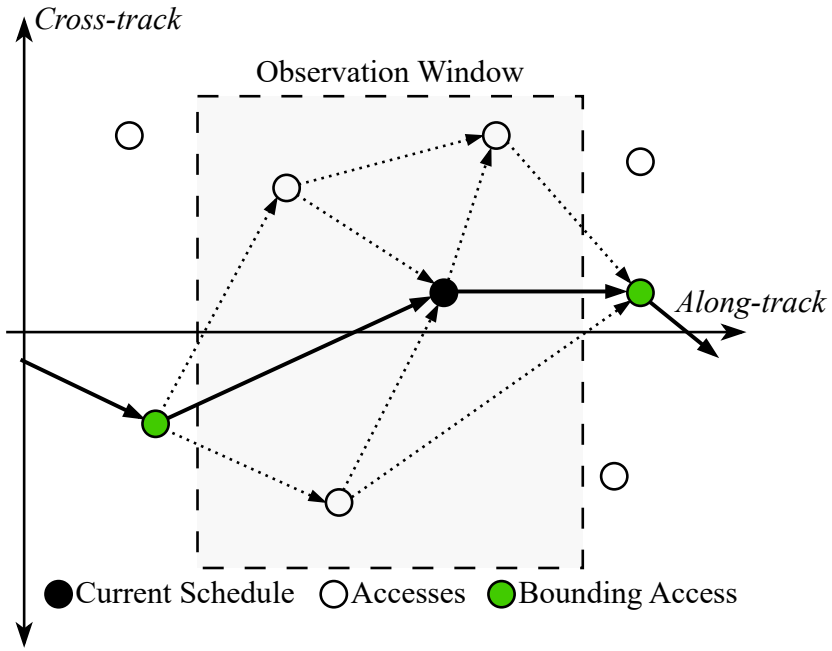


Figure 6.6: Example showing bounding accesses as the closest schedule points outside the observation window.

already in the schedule from an operator perspective, when trying to plan with these actions in the context of a scheduling system, it can lead to having to iterate and figure out which imaging actions are already in the schedule and then backtracking to see if a lookahead action is preferable, adding significant computational complexity.

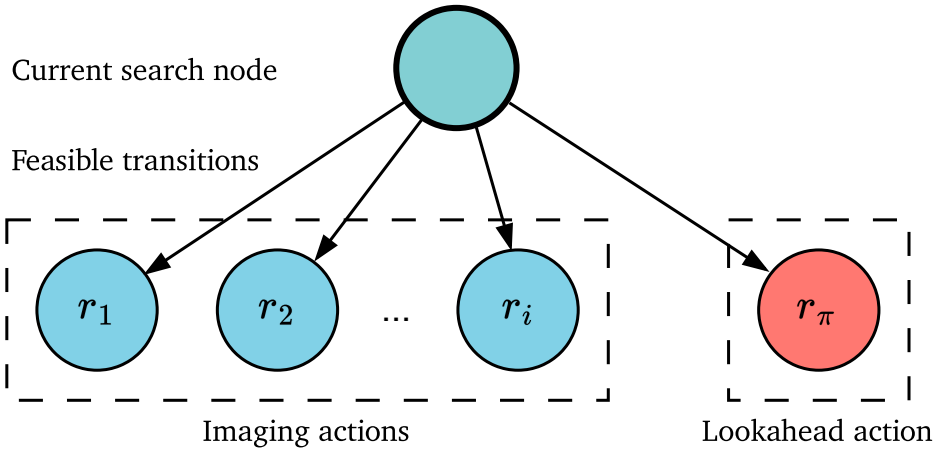


Figure 6.7: Example depicting one-step greedy scheduling with additional lookahead observation incorporated. Depending on the reward r the scheduler could either choose to incorporate either an imaging activity or a lookahead activity.

6.3 Software Requirements

Currently, no existing software package exists that can be used for the breadth of components required for dynamic tasking. The following components have been developed for this purpose, and made as general and as flexible as possible in the `dynamic_tasker.py` Python package.

Figure 6.8: Diagram of `dynamic_tasker.py` software package, showing core modules and functions enclosed within each module.

- **Orbit and Reference-Frame Utilities:** An analytic two-body Keplerian propagator is used, as orbits in this thesis are circular, non-sun-synchronous, and short durations of a maximum of ~24 h. The propagator is configurable and has a backend interface so that alternative engines (e.g. SGP4, Orekit) can be utilized instead. Utility functions to convert between Earth-Centered Inertial (ECI), Earth-Centered Earth-Fixed (ECEF), and latitude/longitude/altitude (LLA) frames are also implemented [119]. The module

also contains helpers for calculating various orbital quantities, such as orbital velocity and period, horizon quantities such as distance and time to the horizon, and projections from ECI points and directions to surface co-ordinates.

- **Dataset and Request Aggregator:** Methods to load different scenarios including the 10,000 world cities dataset, as well as the ability to generate random requests uniformly distributed on Earth's surface.at schedulers remain backend-agnostic.
- **Access Aggregator:** The access aggregation algorithm from Algorithm 1 has been implemented as a singular function, utilizing the aforementioned orbit propagator.
- **Camera Projection and Image Geometry:** Various methods to perform camera projection as described in §6.2.5 Camera Projection and Vision System have been implemented, along with helper methods to clip by z-depth, projection directly from orbit co-ordinates, and unproject back to ECI world co-ordinates.
- **Spherical Geometry and Area Calculation:** Area coverage utilities compute the solid-angle with Gauss-Bonnet of arbitrary vertex sets, for use in footprint coverage analysis, using camera vectors projected onto Earth's surface.
- **Global Cloud-Mask API Hooks:** Helper tools to download, process, and sample data products including cloud masks from the GOES, Meteosat, and Himawari meteorological satellites.
- **Scheduling Engines:** Implementation of MILP and greedy scheduler as described in §4 Satellite Resources, Constraints, and Scheduling.
- **Configuration Constants:** A set of commonly used configuration constants, such as Earth's radius, gravitational parameter, sidereal rotation rate, etc.
- **Dynamic-Tasking Simulation:** A simulation of the spacecraft executing the tasks, combining above components for full, end-to-end dynamic tasking simulation.

Details of the software implementation will not be discussed hereafter. All features are packaged in a Python package called `dynamic_tasker`, with all source code and documentation available at the repository here: https://github.com/Shreeyam/phd_code. All information presented is part of the package and the full source code for core modules is given in §A Core Implementation Details.

6.4 Lookahead Utility

In order to algorithmically determine when and where a lookahead should be performed, the utility of the action must be quantified, in terms of both rewards and costs. Ultimately, the largest reward comes from *schedule improvement*. That is, a lookahead should result in quantifiable increase in the overall schedule according to a chosen metric.

As shown in §3.4 Image Utility Model, in this work we use the total number of cloud-free images as our objective function. Hence, any sensing or lookahead should result in an expected

increase in the total number of cloud-free images in the schedule. In this work, we break this down into two components: schedule advantage, and opportunity cost, as described below.

Finally, once a metric for evaluating schedules is obtained, it can then be used to plan lookahead tasks. The complication with doing this arises due to the continuous nature of lookahead actions in comparison to the discrete and finite nature of individual schedule requests. A lookahead comes with infinitely variable angles and times it can be performed at, and in the most general case, can be done more than once between imaging captures, which may be optimal for certain cases. Evaluating all possible actions leads to a massively high dimensional combinatoric space which is effectively infinite, hence we must limit some aspects in order to make the problem tractable.

6.4.1 Advantage

The schedule advantage can be defined as how much the schedule improves in response to a lookahead, or through any other kind of perception. While this quantity is inherently unknowable, at the minimum its expected value should be greater than zero for any lookahead to be worthwhile on balances.

As the MILP scheduler can find optimal solutions for schedules, the total advantage can be directly obtained in simulation by taking a subset of the omniscient schedule between two specified times, applying agility constraints at the first timestep, and then subtracting the utility of the conventional scheduler evaluated between those two timesteps. In formal terms, the total advantage due to a lookahead at time t, $J_t^+(\boldsymbol{\sigma})$, can be found by the difference in utility of the new re-optimized schedule within the re-optimization window, subtracting the old one, as given by

$$\begin{aligned} J_t^+(\boldsymbol{\sigma}) &= J_t(\boldsymbol{\sigma}_{\text{new}}) - J_t(\boldsymbol{\sigma}), \\ \text{s.t.} \quad & \text{constraints at time } t \end{aligned} \tag{6.8}$$

As aforementioned, since the true value of $J_t^+(\boldsymbol{\sigma})$ is unknowable and highly variable depending on upcoming access states, heuristics must be used to estimate these quantities $\hat{J}_t^+(\boldsymbol{\sigma})$, which is discussed in §6.5 Heuristic Design.

6.4.2 Opportunity Cost

Opportunity cost $J_t^-(\boldsymbol{\sigma})$ due to a lookahead at time t quantifies the cloud-free imaging activities sacrificed when the spacecraft elects to execute a lookahead action instead of continuing with the existing schedule. As this quantity, like the advantage, can only be known in hindsight, it must be estimated by the heuristic also.

The final net schedule advantage can be given by the difference between the schedule advantage and the opportunity cost

$$\Delta J(\boldsymbol{\sigma}) = J_t^+(\boldsymbol{\sigma}) - J_t^-(\boldsymbol{\sigma}). \tag{6.9}$$

The subscript t is omitted on the net schedule advantage, as the quantities for the advantage and opportunity cost are relating to a single lookahead at a particular time, but when combined, the net advantage applies to the whole schedule rather than referring to any individual action.

6.5 Heuristic Design

Now that we have established metrics describing the total utility of a lookahead, we can assemble these pieces together and start constructing heuristics for vision-based dynamic tasking. In this thesis, we go about this in two separate directions: through constructing an analytic heuristic from an approximated version of the scheduling problem whose properties are more tractable to analyze, and through data, using a simulation with synthetically generated data and using machine learning to derive the heuristic for net schedule advantage.

6.5.1 Analytic Heuristic

To derive a closed-form lookahead advantage, we make the following assumptions:

(A1) $\tau_{ij} = \tau_0$, for all task pairs (i, j), so that slewing time between any two accesses is constant;

(A2) $t_i \overset{\text{i.i.d.}}{\sim} \mathcal{U}(T_1, T_2)$, so that accesses are uniformly distributed over $[T_1, T_2]$;

(A3) $X_i \overset{\text{i.i.d.}}{\sim} \text{Bern}(p)$, so the cloud-clear state is a Bernoulli trial with $P(X_i = 1) = p$;

(A4) $c_i = c_0 \; \forall i$, so each task has equal utility.

Under these assumptions, the problem becomes that of finding the expected maximum chain length that can be obtained in the interval $[T_1, T_2]$ with minimum spacing τ_0, given a total number of samples pN of cloud-free accesses.

In order to ease the burden of notation a little, the problem will initially be exclusively referring to the abstract concept of finding chains of length M with minimum spacing g across an interval of length L with N uniformly spaced points, before applying it to the problem of dynamic tasking.

This problem in and of itself is difficult due to the "chain-building" process being inherently non-IID, and due to it itself containing sampling.

While there exists a classic closed-form solution for the probability that N points uniformly distributed within an interval satisfy a minimum gap spacing g that can be obtained using an inclusion-exclusion argument [154]

$$\left(\frac{L - g(M-1)}{L}\right)^M, \tag{6.10}$$

using this same argument for an M-subset of N points to find a maximum spacing likely has

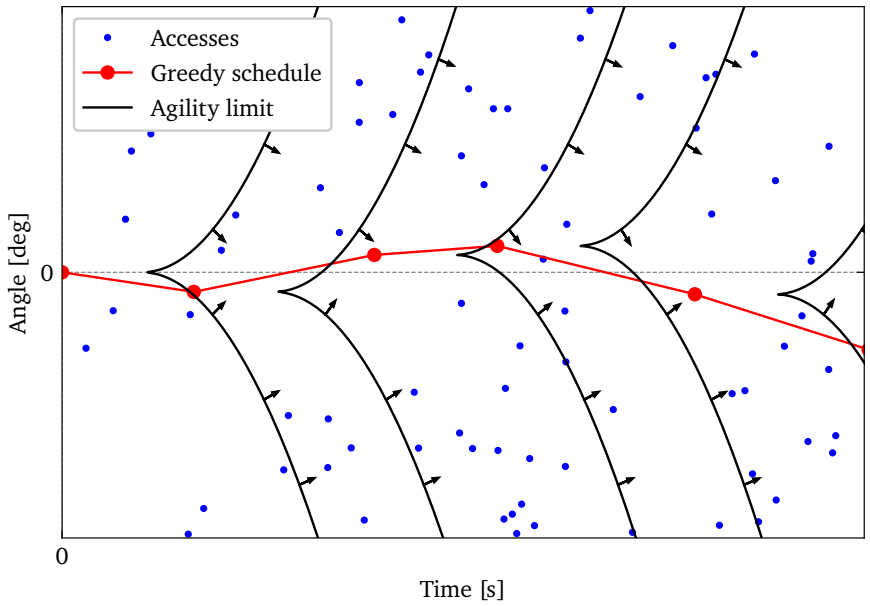


Figure 6.9: Depiction of exact scheduling problem, where the goal is to select the longest weighted path through accesses, subject to constraint manifold formed by the agility function. Example greedy schedule and constraints also overlaid.

no closed form solution using the same arguments, due to the non-IID nature of the points, which is affected by sampling.

Instead, we can use a relationship between a Poisson renewal process and a uniform distribution to approximate this. It turns out that the samples obtained from a Poisson process in a fixed interval conditional on the number of samples are distributed uniformly within that interval. Given this, we can go the other way: given a uniform distribution with a fixed number of samples, we can approximate this as a Poisson renewal process, with the caveat that we are losing the exact fixed-number conditioning when making this approximation.

Given a Poisson renewal process such as

$$R = g + X, \tag{6.11}$$

where X is an exponentially distributed random variable $X \sim \exp(\lambda)$ with a parameter λ, where $\lambda = N/L$. Hence, the expected renewal time (in this case equivalent to mean gap) is therefore

$$\mathbb{E}[R] = g + \frac{1}{\lambda} \tag{6.12}$$

To build an M-length chain requires $M - 1$ gaps in between points. Fixing the first point to the beginning of the interval, the expected value of the process after $M - 1$ renewals is

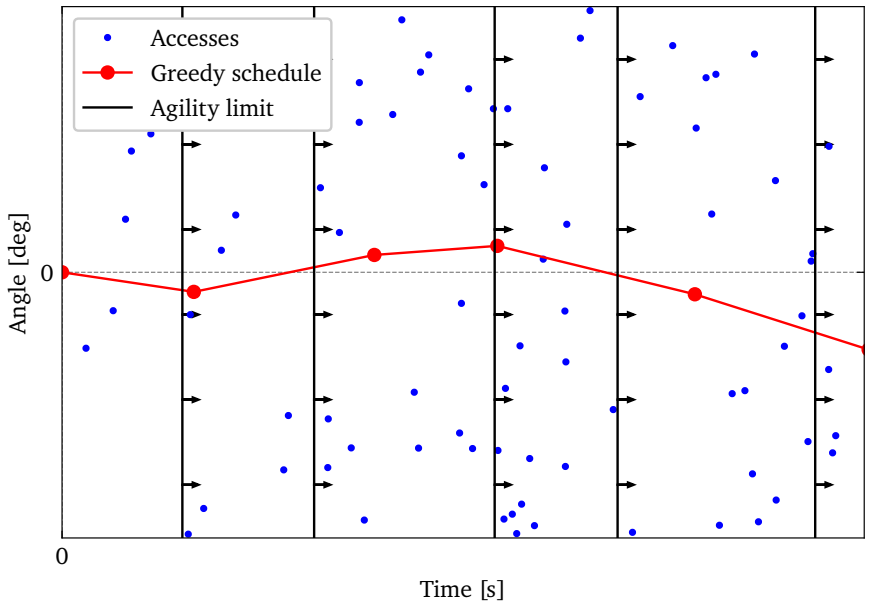


Figure 6.10: Depiction of approximate scheduling problem, where the constraint manifold is uniform, simplifying the problem but adding a parameter for constraint distance.

$$S_{M-1} = \sum_{i=1}^{M-1} R = (M-1)g + \sum_{i=0}^{M-1} X. \tag{6.13}$$

The sum of IID exponential distributions follows a Gamma distribution [155], this can be written using the variable $Y \sim \Gamma(M-1, \lambda)$ as

$$S_{M-1} = \sum_{i=1}^{M-1} R = (M-1)g + Y. \tag{6.14}$$

Now, the probability that the total chain is less than L given by $P(S_{M-1} \leq L)$ is equivalent to

$$P(Y \leq L - (M-1)g), \tag{6.15}$$

hence, the probability that an M-chain exists is

$$G(M; N, L, g) := P(M\text{-chain exists }) \approx \frac{\gamma(M-1, \lambda(L-(M-1)g))}{\Gamma(M-1)} \tag{6.16}$$

where γ is the lower incomplete Gamma function and Γ is the Gamma function. To calculate the expected chain length, it is necessary to remove duplicate entries. The probability that an M length chain exists also depends on the probability that a $M-1, M-2, \ldots, 1$ length chains exist. Hence, the expected value is given by

$$\mathbb{E}[M \mid L, g] = \sum_{m=1}^{\infty} m \cdot (G(m; L, g) - G(m+1; L, g)), \tag{6.17}$$

which through properties of the Gamma function [155] is equivalent to

$$\boxed{\mathbb{E}[M \mid N, L, g] = \sum_{m=1}^{\infty} G(m; N, L, g).} \tag{6.18}$$

In practice, this sum to infinity can be safely truncated once $m \geq L/c + 1$ as the probability drops to zero.

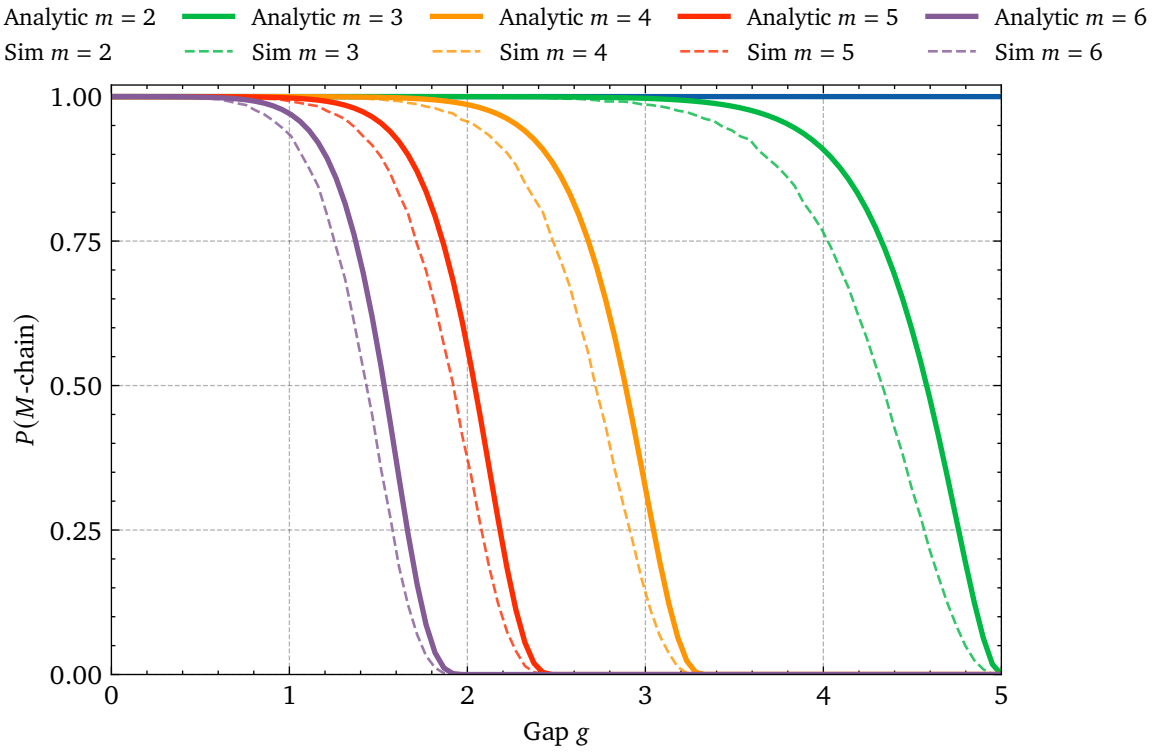


Figure 6.11: Analytic approximation of chain probability in comparison to simulated results. This particular case is of searching for a 4-chain with parameters $N = 20$, $M = 2, 3, 4, 5, 6$, $L = 10$. The analytic solution is slightly more optimistic about finding a chain compared to the simulation in all cases, but retains the correct exact boundary conditions.

While a formal error bound is possible to prove, the expressions and our analysis are sufficiently grounded on simulated results. Figure 6.11 shows the simulated and analytically approximated value of an M-chain existing for values of $M \in \{2, 3, 4, 5, 6\}$ for an interval of size $L = 10$, total number of samples $N = 20$. In all cases, the analytic heuristic is more optimistic about finding a chain, and overestimates the probability that one exists. However, the analytic heuristic gets the boundary conditions exactly correct—for example, finding a 5-chain in an

interval 10 units long any larger than $2\frac{1}{2}$ units is impossible, and is correctly characterized by the analytic estimate.

Now, we can use this analytic approximation in our application case for dynamic tasking. In the spacecraft scheduling problem, we wish to be able to find the expected chain length that can be obtained when observing a set of tasks, as an approximation of how much we can expect our schedule to improve by. In the case of the scheduling problem, a schedule improvement of N requires $N + 2$ additional advantage, due to the bounding accesses around the observation window needing to be in the schedule, adhering to the same gap spacing requirements. While these bounding accesses need not necessarily be in the same interval, in areas with densely packed accesses it is good approximation to roll them into the interval. The true bounding accesses for an observation are shown in Figure 6.6. The gap g then becomes a tunable parameter, essentially denoting how optimistic or pessimistic to be when evaluating how much the spacecraft needs to slew to get to the next task, which can range from t_s up to the maximum slew across the field of regard $t_{\text{slew}}(\theta_{\text{FoR}})$. Since the heuristic as above is optimistic, initial experimentation showed being slightly pessimistic with the gap parameter results in a better advantage estimate. Combining all these aspects together, the analytic advantage estimate given a lookahead at time t, $\hat{J}^{+}_{t,a}$, is given by

$$\boxed{\hat{J}^{+}_{t,a} = \sum_{m=1}^{\infty} G(m+2;\, pN_{\text{obs}},\, \Delta t_{\text{obs}},\, t_g),} \tag{6.19}$$

where p is the probability as defined above of the task being cloud-free, N_{obs} is the total number of accesses observed, $\Delta t_{\text{obs}} = T_2 - T_1$ is the total amount of time in the observation window, and t_g is some expected minimum gap required between tasks. Being pessimistic and assuming a worst-case gap across the field of regard, t_g is hence given by

$$t_g = t_{\text{slew}}(\theta_{\text{FoR}}), \tag{6.20}$$

from which we can now estimate the total advantage of any lookahead action, given the total number of observed points, the total time bounded by the observation window, and the required gap spacing.

We can use the same stationary distribution to estimate the opportunity cost, defining it as

$$\hat{J}^{-}_{t,a}(\boldsymbol{\sigma}) = pN_{\text{missed},\boldsymbol{\sigma}}, \tag{6.21}$$

where $N_{\text{missed},\boldsymbol{\sigma}}$ is the total number of imaging activities in the schedule that were missed due to the slewing time of the activity.

Finally, combining all of these terms together produces an analytically derived estimate of the net schedule advantage for any lookahead action, as given by

$$\widehat{\Delta J}_a = \hat{J}^{+}_{t,a}(\boldsymbol{\sigma}) - \hat{J}^{-}_{t,a}(\boldsymbol{\sigma}). \tag{6.22}$$

6.5.2 Learned Heuristics

Learned heuristics have several potential advantages that rule-based heuristics cannot capture. In particular, learned heuristics can be made very flexible need not be constrained with the assumptions used for the analytic heuristic, such as that of one lookahead per scheduling, equal task utility, equal slewing times, or rectangular lookahead windows. Instead, a machine learning model composed of a neural network can learn the complex heuristic function through either reinforcement learning (RL) or supervised learning, through data generated and sampled with a simulated environment.

Learned policies can also ingest side-information such as location, seasonality, or time-of-day trends, however these factors along with sequenced lookaheads (more than one lookahead per scheduled activity) are out of scope for this work.

Using learned heuristics however also comes with significant challenges both in terms of computation and training. Changes to parameters such as the spacecraft's agility characteristics and lookahead instrument require at least partial re-training to adapt to the new characteristics of the spacecraft, incurring significant computational cost. Reinforcement learning in general also suffers from the problem of non-independent samples: since future samples of the environment are derived from previous actions, they are non-IID, which can cause issues with training stability if a poor action in the past can remove the algorithmic utility of future data. Even simple policy and value functions can take hundreds of thousands of steps to train [156]. In conjunction with the noisy labels resulting from the random access states in the synthetic environment, instability due to non-IID sampling can make RL a non-starter.

On the other hand, using supervised learning necessitates building a large corpus of state, action, and reward triples from simulated data, as well as training time for learning the heuristic from this data.

In this work, as we are training a heuristic considering only locally optimal actions rather than a policy that can run for full episodes, there is no need for training for full episodes, rather only single steps. While this aids verifiability from an operationalization perspective, the fixed steps severely hinder the ability for RL and the ability to bootstrap learning from temporal difference (TD) errors. Hence, while reinforcement learning is likely a promising avenue for dynamic tasking, the format of the problem as presented in this thesis leads us to using supervised learning to generate the heuristic instead.

State and Action Space

The state is an implementation of that described in §6.1 Dynamic Tasking Problem. For an implementation of the heuristic function that can take the state in as an input, the state is required to be of fixed size. This requirement is trivial when including the full state across the entire scheduling block, however when performing trivial optimizations such as culling to

only visible accesses which can vary in number, the state must be transformed so that it can remain fixed-length.

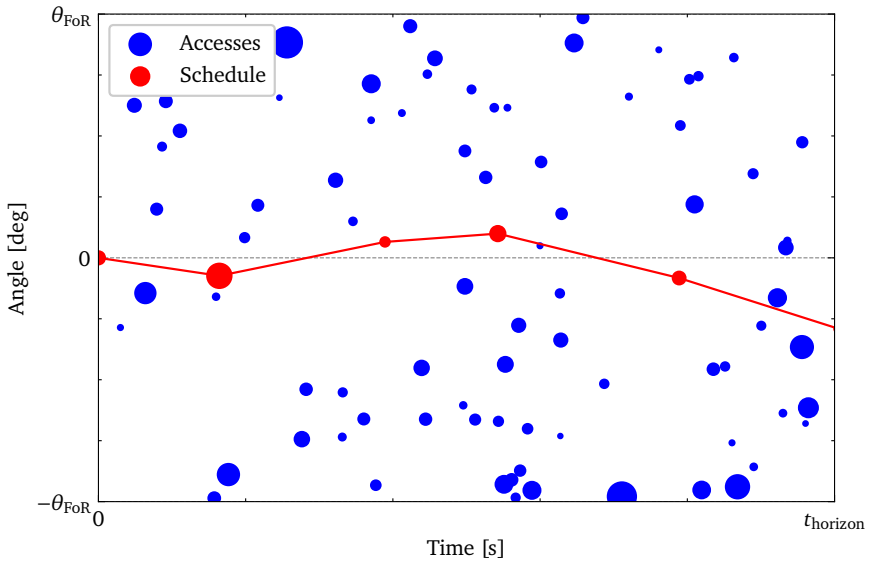
There are many strategies for transforming variable-length vectors into fixed-length representations in machine learning, however rather than using something complex such as an encoder or sequence model, we instead use a two-dimensional histogram-based discretization. The available unobserved accesses, their utility, as well as the current schedule and their utility values are transformed in terms of time and roll angle and summed into bins, up to the observable (horizon) distance and field of regard limits. While this method incorporates additional parameters in terms of discretization resolution and can be inefficient from a memory standpoint, it is the simplest to implement, can be debugged easily, and from initial testing, even a coarse discretization is sufficient. The final state vector has four channels, consisting of the two-dimensional histogram of accesses, access utility, the current schedule, and schedule utility, all taken from the current time up to the horizon, and including both extremes in the cross-track direction, up to the spacecraft's field of regard. A diagram of how this technique works is shown in Figure 6.12, showing how the current accesses and schedule state within the horizon window are decomposed into the four channels.

Simulation Environment and Data Generation

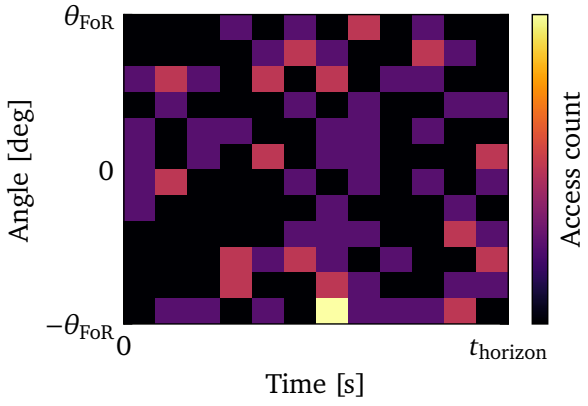
The environment used for this work is shown in Figure 6.13. To make the software implementation as general as possible, the environment has been implemented as a *reinforcement learning* style gym environment, but will be used to generate simulated data for a heuristic trained through *supervised learning*.

The agent first picks an action a_t, consisting of the pitch and roll angle of the lookahead. This information is then fed to three separate locations to return the current reward and updated state:

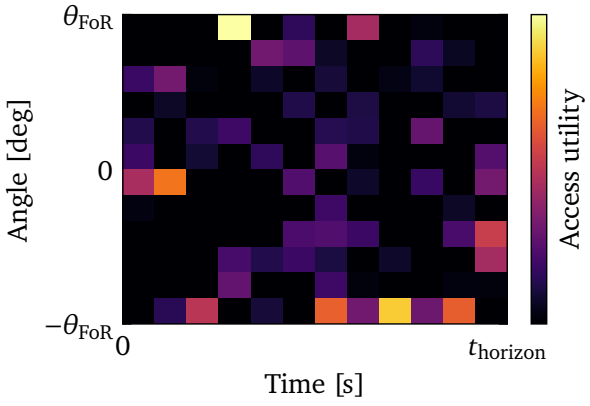
- **Opportunity Cost:** Based on the action, the total maneuver time is calculated from the agility function, based on the worst case of slewing to that attitude and back to the neutral position. Then, the environment calculates the number of tasks in the schedule that were non-cloudy that the spacecraft missed, and sums their utility to derive the final opportunity cost.
- **Advantage Calculation:** After the lookahead action is performed, imaging accesses are then sent through the camera simulation, and the onboard estimate of access utility is updated. Then, based on these updated utility values, the schedule slice between the bounding accesses as shown in Figure 6.6, and is re-optimized using a MILP scheduler, and the advantage is calculated as the increase in total utility between the re-optimized and original schedules.
- **Orbit Propagation and State Update:** Based on the lookahead action and camera projection, the state inside the environment is updated, with tasks marked as observed and their updated utility values now known to the agent. The environment is then incremented to the next scheduling activity via the orbit propagator, and the updated state is returned.



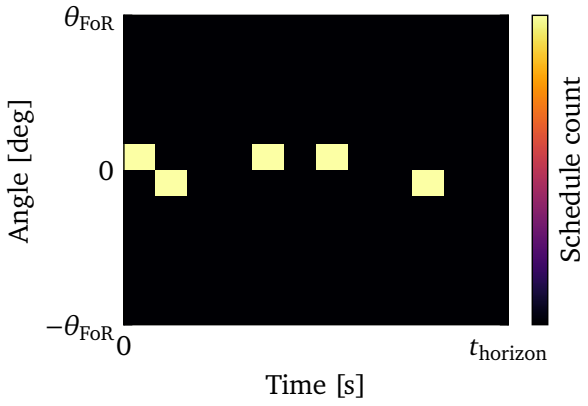
(a) Representation of current state up to horizon, showing accesses in blue sized by utility, and current schedule in red.



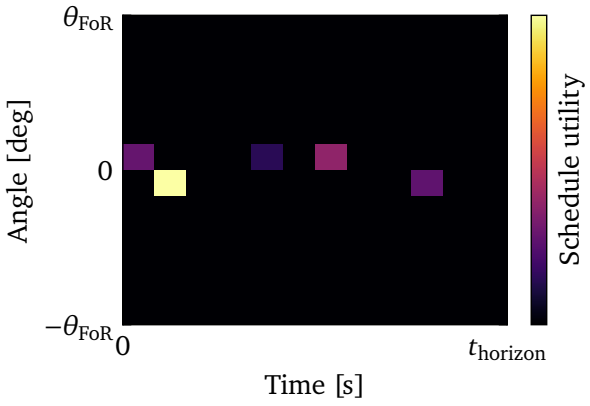
(b) Access quantity.



(c) Access utility.



(d) Schedule quantity.



(e) Schedule utility.

Figure 6.12: Diagram of four-channel histogramming approach, used to convert variable length number of accesses within scheduling horizon into a fixed-dimension multichannel representation for input to the heuristic model. Colorbar labels and precise axes ticks are omitted as this is purely a representative diagram.

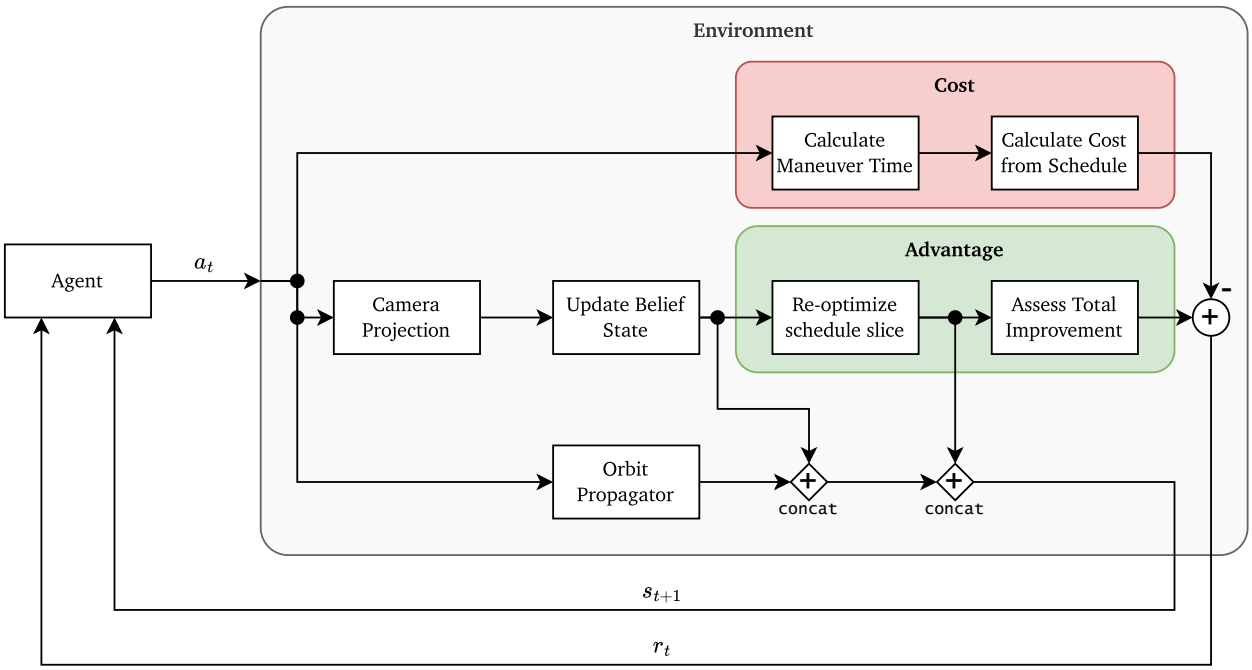


Figure 6.13: State, environment, and reward structure for training lookahead heuristic. Full software pipeline is used to simulate vision instrument, obtain updated access utilities based on capture, and reschedule based on updated utility values. Re-optimized schedule advantage is calculated and opportunity cost based on missed opportunities is subtracted, to generate final reward for the agent.

The reward function is simply returned as the raw advantage subtracting the opportunity cost. While there are many possible avenues for reward shaping, this particular reward was chosen since it is well-posed, stationary, and is the metric that we wish for a heuristic to return when deploying onboard.

Unlike the analytic heuristic, since all of these actions happen inside a simulator where all quantities are known to the environment (but not the agent), true values of opportunity cost and lookahead advantage can be used, potentially allowing the learned heuristic to pick up more subtle dependencies of the advantage with the spacecraft state that the analytic heuristic had to purely estimate.

In the supervised learning case, random pitch and roll actions are fed into the environment, generating triples of the state, action, and net schedule advantage. 1 200 000 triples were generated using this method, to create a corpus for training the heuristic. While this is a large dataset, early empirical testing showed a significant amount of delayed generalization when training on a dataset 10% of the size [157], which was ameliorated with significantly increasing dataset size [158]. As many training samples and variation are required for training the heuristic, rather than training on the 10,000 world cities dataset which could introduce some bias, 30,000 random global points (to account for global land/water coverage) along with random access states and random Pareto-distributed task utilities are used to aid in generalization. Figure 6.14 shows this concept graphically—that a random environment can

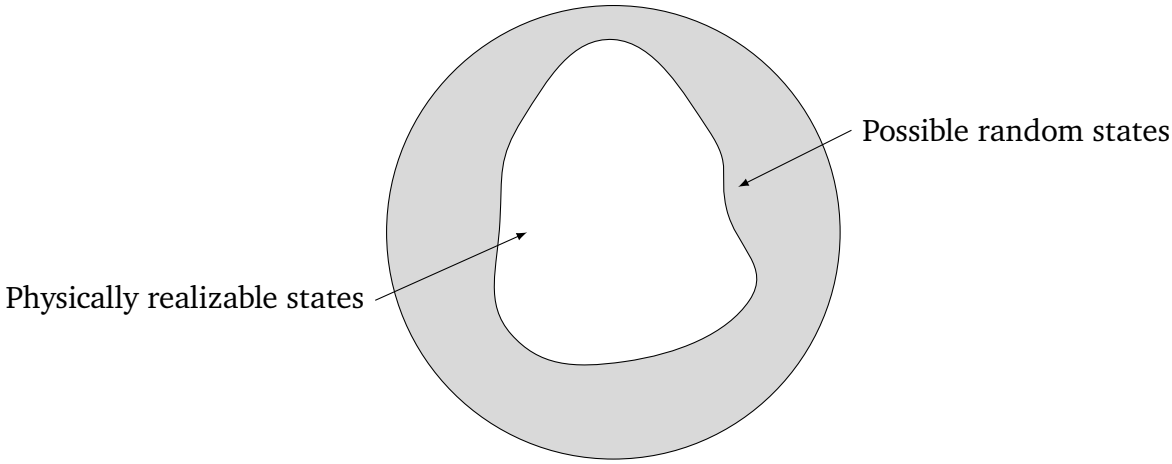


Figure 6.14: Diagram showing how training on randomized data can generalize to situations found in the physical world, as the set of random states is larger than the set of physically realizeable states.

superset the physically realizable states.

While the random scenario generation in the environment can simulate all possible states, noisy labels combined with under-representation of states that are similar to physical ones can result in out-of-distribution behavior when training on synthetic data and deploying the model in a realistic simulation scenario. In particular, the 10,000 world cities dataset is significantly *higher entropy* due to clustering than the synthetic data. For instance, the amount of water mass on Earth results in the 10,000 world cities dataset having large swaths where the state is purely zero, a scenario unlikely to be found in the simulated data with randomly distributed points. To augment the synthetic data to match closer to the actual test distribution of 10,000 world cities, with a random probability the simulation environment will mark all accesses up to the horizon as observed and re-optimize the schedule, indicating no further improvement is possible, corresponding to areas such as deserts and oceans in the test dataset.

Model Design and Training

Figure 6.15 shows the architecture of the machine learning model used for the heuristic in this work. The heuristic is likely to be close to linear due to the connection between coverage and lookahead pitch up to the horizon, motivating a simple architecture, with a limited number of layers. As the state input through the histogramming procedure as described in §6.5.2 State and Action Space is a multichannel, two-dimensional image-like input, convolutional layers are used to extract spatial features, such as those relating to agility constraints. The size of the convolutional kernel is chosen based on modern approaches for improving performance of convolutional neural networks [159]. Then, the action vectors are concatenated to the convolutional features, and the entire vector is sent through a multi-layer perceptron, outputting the final net schedule advantage value.

Various hyperparameters are tuned based on performance on the validation set, which are shown in Table 6.1. All model training and inference are conducted in PyTorch. For training,

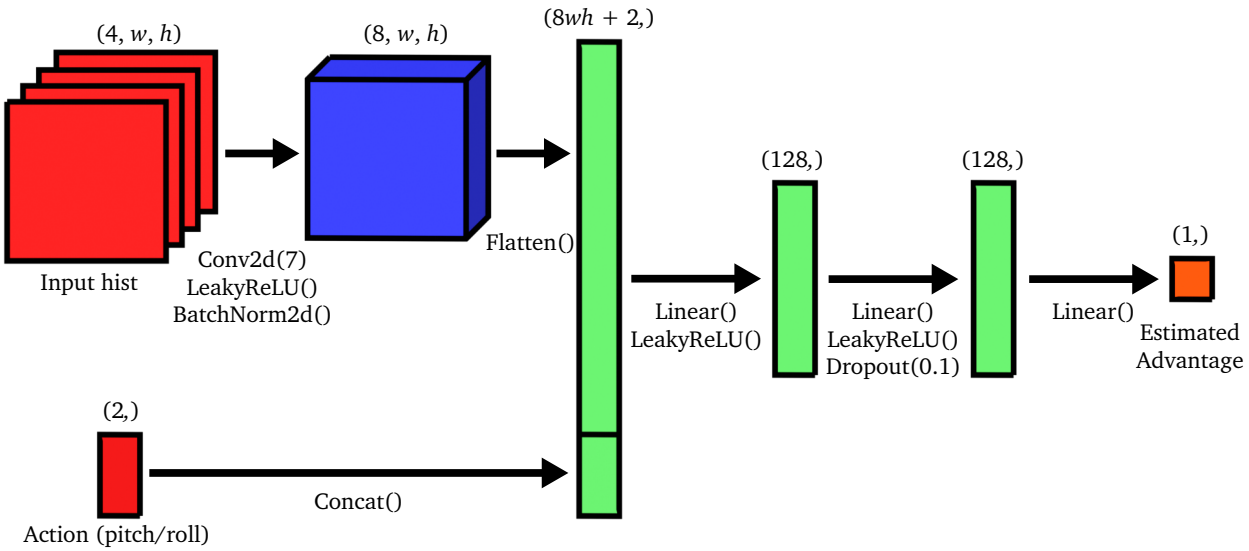


Figure 6.15: Diagram of model used to transform state and action inputs into single scalar heuristic describing net schedule advantage. Four-channel input state (discretization is not specified so w and h are used as placeholders) is run through a single convolutional layer to extract features, flattened and concatenated with the action input. A small multi-layer perceptron (MLP) trained with dropout is then used to estimate the final advantage. Input layers are red, convolutional layers are blue, MLP layers are green, and outputs are orange in this diagram.

the 1.2 million (s, a, r) triples as described in §6.5.2 Simulation Environment and Data Generation are used, with 40% of the dataset consisting of null (already fully observed with no possible advantage) data, and split 80% and 20% between a training and validation set (the actual test set consists of a full dynamic tasking simulation with realistic task states and utility values). Dataset size is chosen based on initial experimentation; smaller dataset sizes of 120,000 triples showed a large amount of delayed generalization, motivating significantly higher amounts of synthetic data [157, 158]. Additionally, the long-tailed Pareto distribution for task utility similarly motivates an extremely large dataset, to adequately characterize large, infrequent task utilities.

Schedule advantage is based on randomized task states as well as the long-tailed Pareto-distrbiuted task utility, resulting in extremely noisy labels. When used in conjunction with a loss function like mean squared error (MSE) loss, gradients can become extremely large and caused significant training instability in initial experimentation. To mitigate the destabilizing influence of these large gradients, we instead adopt L1 loss for our loss function, otherwise known as mean absolute error (MAE), which is more robust to outliers and noisy labels.

6.6 Results

For our results we first dissect individual components of the approach, such as the predictive power of cloud masks as priors and training the learned heuristics, and then incorporating

Table 6.1: Summary of hyperparameters used to train learned heuristic, along with dataset parameters.

Hyperparameter	Value
Learning rate	1×10^{-4}
Batch size	256
Epochs	15
Optimizer	AdamW
AdamW β_1	0.9
AdamW β_2	0.999
Weight decay	0.01
Loss function	L1 Loss
Dataset size	1.2×10^6
Data null rate	40%
Validation split	20%

them into dynamic tasking algorithms.

We split results for the variations on the dynamic tasking algorithms as well, by first using individual regional analyses for specific cases, and then aggregating metrics over a long scheduling block.

6.6.1 Analysis Cases

Prior chapters in this thesis have produced cases for spacecraft agility, instrumentation, and heuristics based on specific analyses. All of the cases considered for this work are shown in Table 6.2 and Table 6.3, split up due to the different paradigm involved.

Table 6.2: Comparison cases for vision-based dynamic tasking in this work, showing permutations of spacecraft agility class (Table 3.3), instrument classes (Table 5.1), and lookahead heuristics for agile lookahead.

Case	Agility Model	Lookahead Instrument	Lookahead Heuristic
1	Agile	Conventional (None)	N/A
2	Agile	Omniscient (None)	N/A
3	Agile	60°×60°	N/A
4	Agile	45°×45°	Analytic
5	Agile	45°×45°	Learned
6	Ultra-agile	Conventional (None)	N/A
7	Ultra-agile	Omniscient (None)	N/A
8	Ultra-agile	60°×60°	N/A
9	Ultra-agile	45°×45°	Analytic
10	Ultra-agile	45°×45°	Learned

Table 6.3: Comparison cases for dynamic tasking based on meteorological data used in this work, showing permutations of spacecraft agility class (Table 3.3), intrinsic information delay (Table 5.1), and link between meteorological satellite and imaging satellite.

Case	Agility Model	Information Delay	Link Type
1	Agile	Conventional (N/A)	N/A
2	Agile	Omniscient (N/A)	N/A
3	Agile	{0, 30, 60, 90, 120} minutes	ISL (Instant)
4	Agile	{0, 30, 60, 90, 120} minutes	Ground station (Cyclic)
1	Ultra-agile	Conventional (N/A)	N/A
2	Ultra-agile	Omniscient (N/A)	N/A
3	Ultra-agile	{0, 30, 60, 90, 120} minutes	ISL (Instant)
4	Ultra-agile	{0, 30, 60, 90, 120} minutes	Ground station (Cyclic)

Table 6.4: Description of synthetic and realistic environments used in this work. The synthetic environment is used to generate scenarios for training and algorithmic comparisons, as it provides an unbiased scenario that can be re-generated indefinitely for generalization and comparison, and the realistic environment is used for real-world operational evaluation.

Environment	Requests	States	Utility	Purpose
Synthetic	30,000 random	Weighted random	Pareto	Training
Eval	10,000 world cities	Weighted random	Unit	Evaluation
Realistic	10,000 world cities	Meteorological data	Pareto	Evaluation

6.6.2 Predictive Power of Cloud Mask Priors

Before presenting the results from the dynamic tasking algorithm utilizing meteorological lookahead, we can first assess the predictive ability of cloud masks as priors with real data, to obtain an overview of raw predictive power, unbiased from downstream sampling effects from scheduling and dynamic tasking.

Figure 6.16 shows the probability of a later binarized cloud mask being equal to a prior. The probability drops off rapidly, before asymptoting the expected values of approximately 66% cloud cover. Figure 6.17 shows the Bayes factor of using a previous cloud mask as a prior up to 35.5 hours, compared to the naive prior used in conventional satellite scheduling where all tasks are assumed cloud-free. Initially the mask-based prior has significantly more predictive power than the naive, before dropping down and asymptoting at around twice the power. This behavior is likely due to individual clouds being hard to predict, but their formation conditions e.g. humid updrafts near mountains being relatively stagnant, resulting in aggregate higher predictive power.

Figure 6.18 shows the same data, but now as a locus on a precision-recall plot. With the assumption that baseline predictive power is achieved after 35.5 hours reaching, precision and recall appear to decay linearly to an ultimate value of approximately (0.73, 0.77), showing

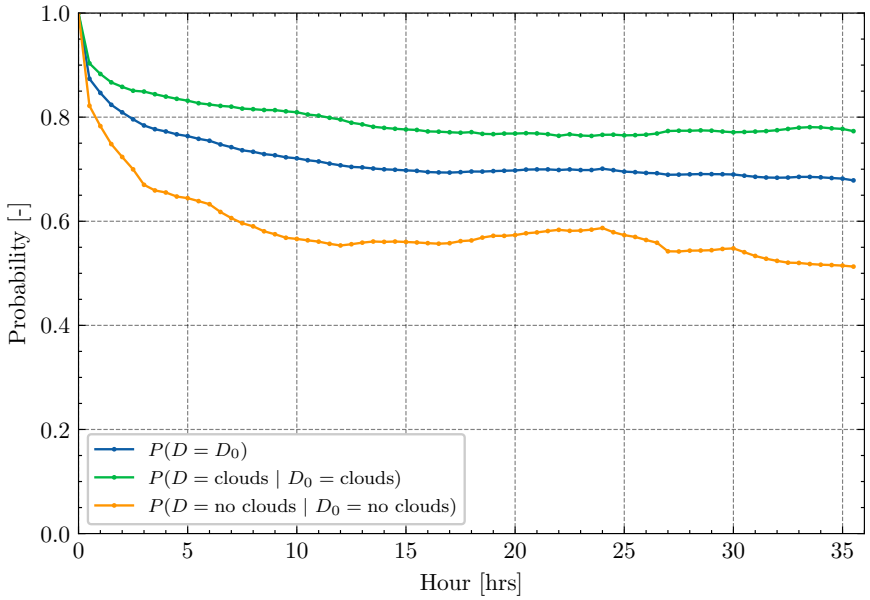


Figure 6.16: Probability of cloud status being predicted by a previous cloud mask from the past, split by total, and initial classification. Data is taken from a global cloud mask at 2025-01-01T00:00:00+00:00 looking up to 35 hours in the future.

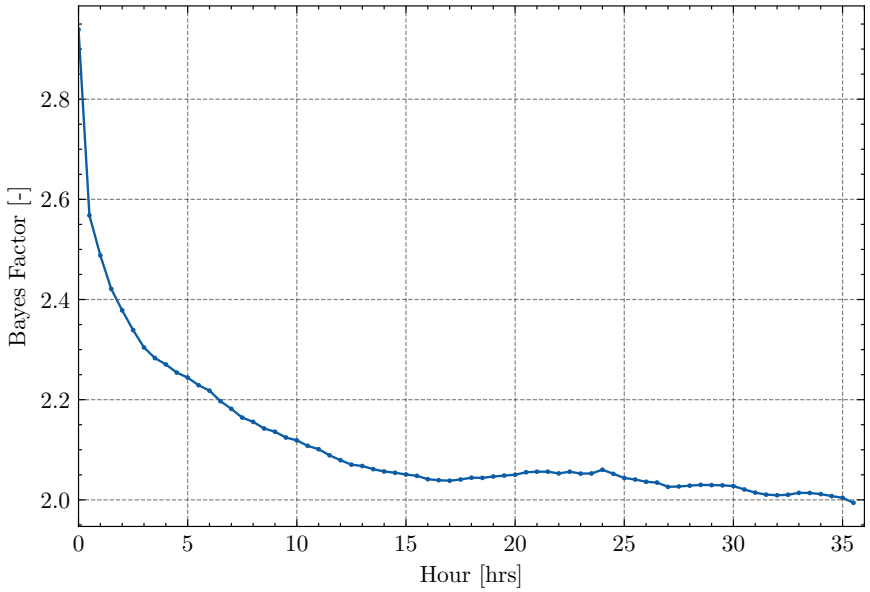


Figure 6.17: Bayes factor of comparing a prior equal to an earlier observation with a naive prior that all tasks are always cloud-free, showing significantly higher predictive ability of the earlier observation compared to the naive model. The Bayes factor initially drops off rapidly before asymptoting around a value of 2.0, showing that latency is a big factor in predictive ability. Data is taken from a global cloud mask at 2025-01-01T00:00:00+00:00 looking up to 35.5 hours in the future.

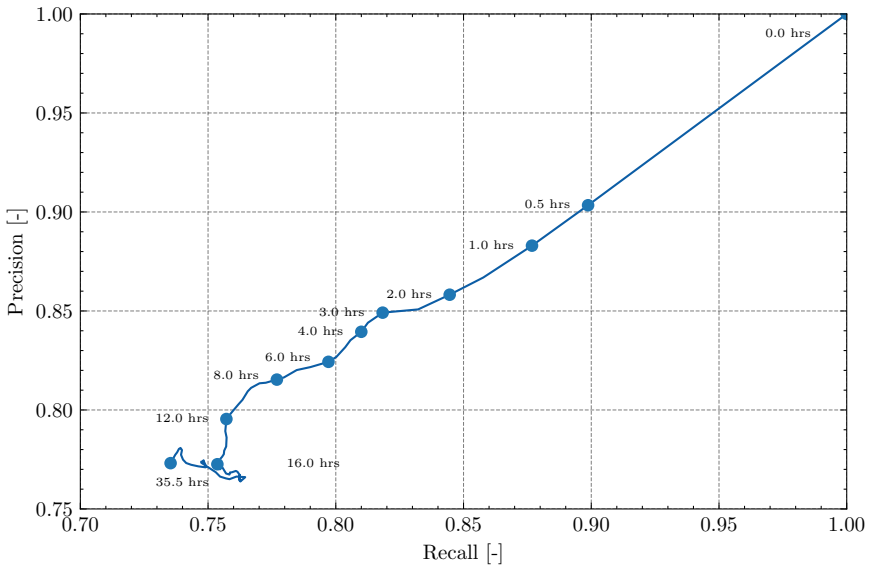


Figure 6.18: Precision-recall curve for cloud prediction from prior masks with a time delay, showing diminishing predictive ability of cloud mask priors with time. Data is taken from a global cloud mask at 2025-01-01T00:00:00+00:00 looking up to 35.5 hours in the future. Within two hours, most of the predictive power compared to the baseline is lost.

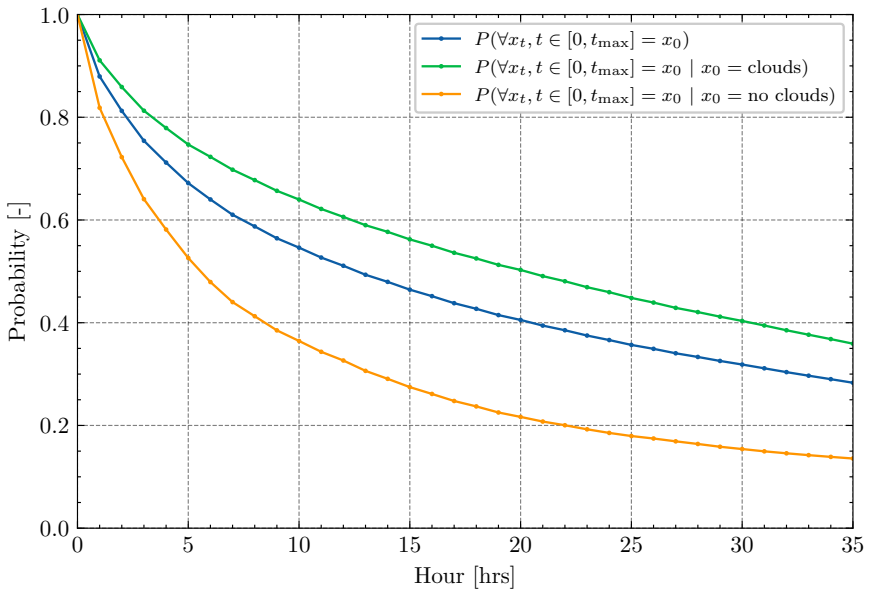


Figure 6.19: Probability of a sampled point remaining the same state as its initial state. Data is taken from a global cloud mask at 2025-01-01T00:00:00+00:00 looking up to 35.5 hours in the future. While the predictive power of an initial cloud mask levels off, it does not mean that each individual sample remains unchanging, as evidenced by the exponential drop in the probability shown in this plot.

Table 6.5: Summary of all parameters used in dynamic tasking simulation.

Parameter	Symbol	Value
Schedule Block Start		2025-01-01T00:00:00+00
Schedule Block End		2025-01-02T00:00:00+00
Schedule Horizon		12 hours
Orbit Inclination	i	51.6°
Orbit Altitude	h	400 km
Orbit Period	T	5553.5 s

no strong class bias. Predictive power also reaches half its baseline value after two hours, emphasizing the importance of timeliness in using prior masks.

6.6.3 Learned Heuristic Training

Figure 6.20 shows the loss function over 15 epochs of training for the agile and ultra-agile heuristic. While the regularization and previous tuning helps the heuristic to generalize, learning the full long tail of the Pareto-distributed tasks without overfitting on the data can still be difficult, leading to an initial drop in test loss followed by a plateau around 8 epochs, while train loss continues to decrease. Additionally, the extremely noisy labels cause the loss to plateau at a high nonzero value, as the underlying features are purely stochastic.

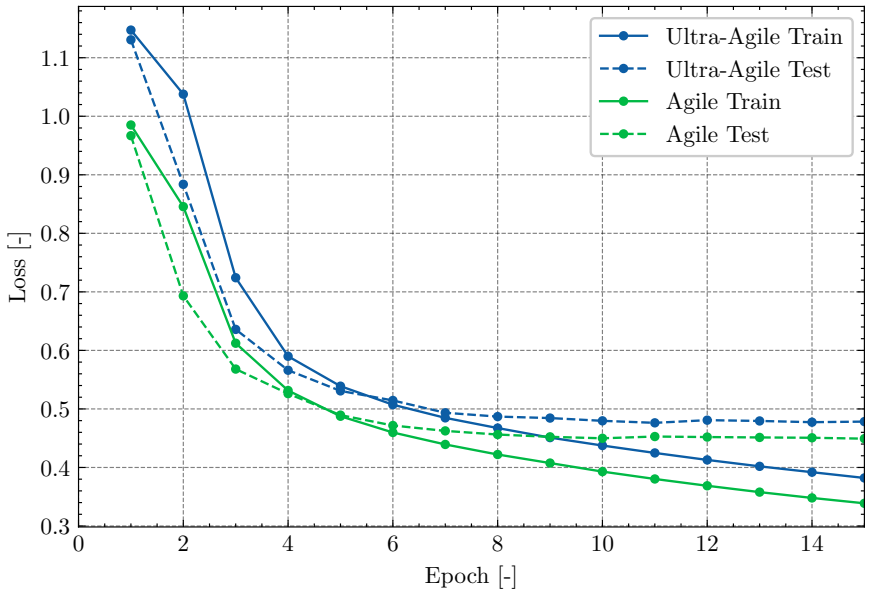


Figure 6.20: Training and test L1 loss after training for 15 epochs. Initially test and train loss decrease together, indicating generalization across both sets, but training is terminated after 50 epochs to prevent overfitting on training data.

6.6.4 Unit Utility Case (Vision Only)

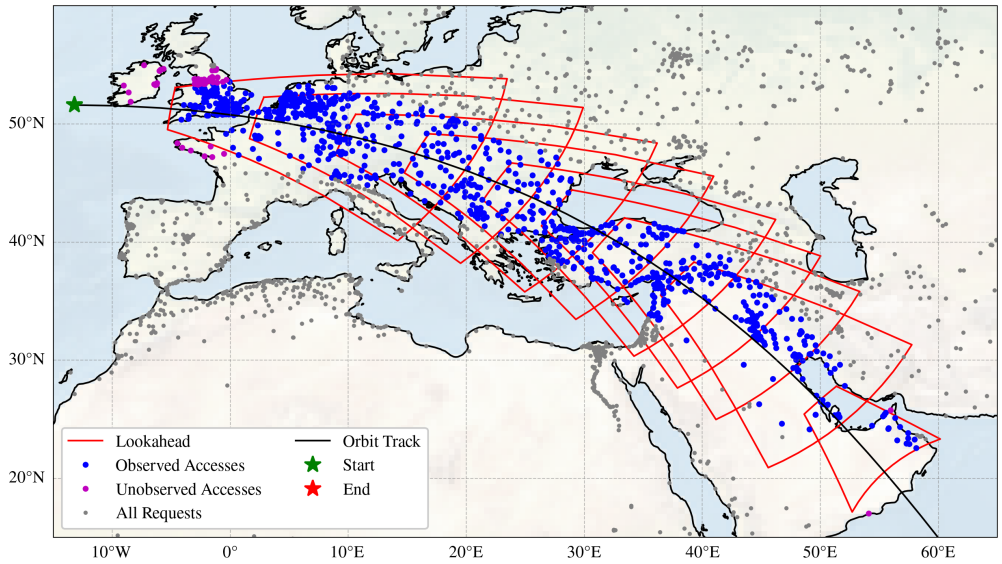


Figure 6.21: Example single-satellite dynamic tasking simulation run over Europe. Lookaheads are primarily looking at the horizon to maximize upcoming task coverage, excluding areas at the beginning of the schedule left unobserved due to insufficient advantage, and a smaller lookahead at the very end due to no further imaging accesses.

We use an orbit track over Europe consisting of densely packed imaging accesses, which is likely the most challenging case for trading off between densely packed imaging accesses and performing lookaheads, as there is limited agility to perform both. Figure 6.22 shows a trial over Europe for the analytic heuristic. The orbit starts west of the coast of Ireland, where the analytic heuristic shows no initial gain in performing a lookahead and so leaves some initial tasks unobserved, before moving to continental Europe, where there is a consistent cadence of lookaheads to optimize the schedule, with more lookaheads in areas with denser tasks and higher possible advantage. As the number of available accesses drops over the Middle East, one final, small lookahead is performed as there is no advantage to slewing further to observe the final unobserved task. This behavior shows that in this case where tasks are densely populated, the heuristic estimates that the best local actions are to pitch as close as possible to cover the horizon, but is still able to adapt to conditions where there the marginal utility of a lookahead close to the horizon is zero. Since in this case, a lookahead is performed greedily every time the heuristic estimates the action to have a positive outcome, more lookahead are performed than necessary, resulting in dense covers of upcoming imaging accesses.

Figure 6.22 shows the same orbital track but using the learned heuristic instead. The behavior is broadly similar, choosing to perform large lookahead actions close to the horizon. Initial schedule performance is nearly identical, with the learned heuristic also choosing to forego a lookahead on the first set of imaging activities, before doing a broad lookahead over areas of western Europe. The learned heuristic appears to be more sensitive to opportunity costs, choosing not to perform additional lookahead in the most target-dense areas, then maintains a steady cadence of lookaheads thereafter.

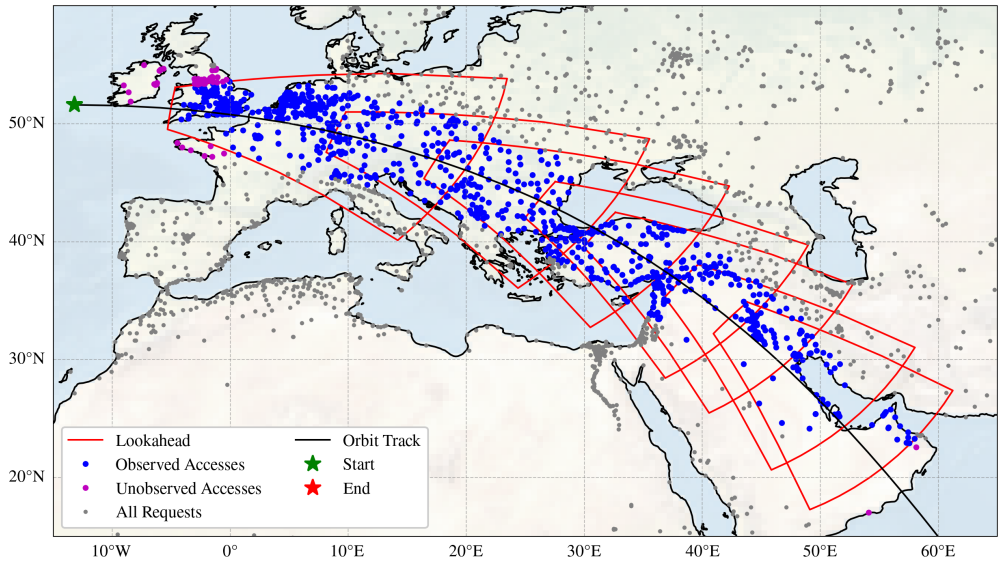


Figure 6.22: Example single-satellite dynamic tasking simulation run over Europe. Lookaheads are primarily looking at the horizon to maximize upcoming task coverage, excluding areas at the beginning of the schedule left unobserved due to insufficient advantage, and a smaller lookahead at the very end due to no further imaging accesses.

Table 6.6 shows all of the considered algorithms for each agility case for 20 trials of a full 24-hour scheduling run with parameters given in Table 6.5, along with the conventional and omniscient cases for comparison. Compared to the conventional scheduler, the omniscient case produces a schedule with significantly fewer tasks, implying that there are periods where the spacecraft sits idle due to insufficient supply of cloud-free imaging activities. For the agile case, the omniscient schedule is on average 30% shorter for the agile case and 33% shorter for the ultra-agile case, but with approximately double the number of cloud-free captures in both cases.

Overall, the stationary heuristic with the wide field of view instrument does the best in terms of overall utility—not requiring any slewing actions for full observability comes with only the downside of local, and sometimes last-minute planning requirements. In the stationary case, the schedules produced are on average 97.1% the utility of the omniscient scheduler in the agile case and 98.3% in the ultra-agile case.

In the agile case, the analytic and learned heuristic are comparable in performance, with an average schedule utility of 353.1 and 355.1 respectively, both approximately 70% of the omniscient utility, and both over 40% better than the conventional scheduler utility, showing major increases in useful image throughput. Both perform approximately 120-130 lookaheads in the schedule, with the learned heuristic performing slightly fewer, as previously observed. Schedule improvement per lookahead is on average 0.83 and 0.87 for the analytic and learned cases respectively. The estimated schedule improvement is 1.59, which could imply some amount of overestimation of lookahead utility, but is likely an artifact of the biased sampling caused by lookaheads only being taken if the estimated utility is greater than zero. Despite efforts to mitigate distribution shfit between the synthetic validation and the 10,000 world

cities test datasets, there still appears to be some distribution shift skewing the data, where lookaheads over densely populated regions such as western Europe can cause weights in the model to be overwhelmed estimating utility values as high as 40, even though in actuality the maximum schedule utility is only around 7-8 cloud-free accesses.

Across all 20 trials the lookahead cases perform better than the conventional cases, showing that even in the agile cases where there are tradeoffs in performing lookaheads, that the heuristics developed in this thesis are robust enough to take those into account and deliver superior throughput of useful imaging data, built from existing schedules already in use for spacecraft tasking and scheduling.

6.6.5 Realistic Case (Pareto Utility)

Now we consider the realistic cases, where meteorological data is used as ground truth and the tasks are weighted randomly according to a Pareto distribution, including also using meteorological data for lookahead, with both ISLs and scheduling through ground stations. These lookahead modalities are separated into two separate tables for the agility cases considered, in Table 6.7 and Table 6.8 for the agile and ultra-agile agility cases, respectively.

First we analyze the conventional and omniscient cases to determine how much the Pareto distribution alone affects the fundamental scheduling problem. Utility values in this case are decoupled from the schedule length due to the long-tailed distribution. The length of the conventional schedule is decreased significantly from 717.0 to 558.4 for the agile case and 1077.0 to 878.4 for the agile and ultra-agile cases respectively, representing 22% and 18% drop. Since the utility values of each task can vary so much, likely what's happening is that the scheduler is preferentially taking tasks of higher utility and more on the tail end of the utility distribution, as the tradeoff with the increased agility becomes more worth it. The same trend can be seen for the omniscient schedule, hinting at the same trade-off of shorter schedules with larger utility values: a decrease in schedule length from 505.3 to 431.4 (-15%) and 720.7 to 646.7 (-10%) for the agile and ultra-agile cases respectively.

Another striking difference between the unit utility case is that the conventional and omniscient schedule are significantly closer in utility, with the omniscient scheduler producing a schedule with only 4.4% more utility in the agile case and 3.6% in the ultra-agile case. This meager improvement is likely caused by interaction of the Pareto-distributed task utility and the underlying orienteering problem—the gap in utility between regular tasks and high-value tasks is so great and long tailed, that even in cases where a majority of access are cloudy, they are still worth taking as the expected value is likely still significantly larger than swapping any others in the schedule, echoing a fundamental property of the Pareto distribution, that a large amount of the utility can be obtained with a vast minority of the accesses, mitigating the effect of cloudy activities altogether. This property interacts with the stochastic orienteering problem in the sense that even if the conventional schedule diverges from the omniscient schedule in some locations, there are still likely other high-value alternatives that can be taken even if not the exact optimal case. In light of this collapsing differential between the omniscient and conventional scheduler, improvements become much more challenging to obtain—any dynamic tasking algorithm has relatively little to gain but much more to lose,

Table 6.6: Table of results for simulation of agile and ultra-agile cases over unit utility environment, showing agility case, lookahead heuristic, initial and final schedule lengths $|\boldsymbol{\sigma}|$, initial and final schedule utilities $J_{\boldsymbol{\sigma}}$, number of lookaheads $N_{\text{lookaheads}}$, and improvement per lookahead $\Delta J / N_{\text{lookaheads}}$.

Agility	**Case**	**Heuristic**	$\lvert\boldsymbol{\sigma}_{\text{final}}\rvert$	$J_{\boldsymbol{\sigma}_{\text{final}}}$	$N_{\text{lookaheads}}$	$\Delta J / N_{\text{lookaheads}}$	$\widehat{\Delta J}$
Agile	Conventional		717.0 (717–717)	247.2 (229–274)			
Agile	Omniscient		505.3 (495–517)	505.3 (495–517)			
Agile	Lookahead		509.2 (491–530)	490.7 (472–511)			
Agile	Lookahead		424.6 (415–435)	353.1 (339–370)	127.2 (121–133)	0.83 (-8.0–10.0)	1.59 (0.0–7.3)
Agile	Lookahead		431.1 (420–444)	355.1 (341–380)	124.5 (117–130)	0.87 (-5.0–10.0)	2.2 (0.0–40.1)
Ultra-agile	Conventional		1077.0 (1077–1077)	365.6 (334–386)			
Ultra-agile	Omniscient		720.65 (694–746)	720.65 (694–746)			
Ultra-agile	Lookahead	Stationary	726.1 (703–749)	708.2 (686–729)			
Ultra-agile	Lookahead	Analytic	521.2 (495–538)	468.7 (447–494)	224.85 (218–232)	0.46 (-10.0–16.0)	1.38 (0.0–13.1)
Ultra-agile	Lookahead	Learned	666.9 (627–687)	562.8 (527–590)	115.25 (111–119)	1.71 (-4.0–16.0)	1.27 (0.0–8.8)

making balancing between opportunity cost and schedule advantage much more sensitive.

Moving on towards analyzing the dynamic tasking algorithms building upon the conventional schedule, in the agile case, only dynamic tasking with stationary lookaheads and agile lookahead with the learned heuristic can improve on the performance of the conventional scheduler, with 2.9% and 0.05% increase in schedule utility, respectively. While the improvement was found over twenty scheduling runs, the tiny increase in utility for the learned heuristic is liable to evaporate when considering increased power requirements for agile lookahead as compared to the stationary or conventional case. In the ultra-agile case, the same trend can be observed even more pronounced, with only the stationary lookahead doing better than the conventional method. None of the cases using meteorological data show improvement with the delay times considered, however in a theoretical case with an ISL and no data delay, the results would be comparable to the stationary lookahead case. Given that meteorological data from the satellites considered is processed and publicly available on AWS S3 within minutes after downlink, the awareness and latency are comparable to the stationary lookahead case and would match or exceed its performance, as the stationary lookahead is limited to a smaller scheduling horizon than global awareness would give.

For both the agile and ultra-agile cases the analytic heuristic performs identically in behavior to the unit utility environment, which is extremely detrimental when the utilities are Pareto-distributed—in the agile case, it loses 33.6% of the utility compared to the conventional scheduler, and in the ultra-agile case, 34.5%. Since the analytic heuristic fundamentally assumes unit utility, there is no alignment between the true and estimated advantage when run in the Pareto case. This becomes even more apparent when observing the difference between the estimated and actual utility of each lookahead: while the analytic heuristic estimates a net schedule advantage of 0 - 13.1 per each lookahead taken with an average of 1.59, the actual net schedule advantage varies between -13833.5 - 733.5 with an average net loss in schedule utility of -50.5 per lookahead. Fundamentally the analytic heuristic can only be constructed assuming net utility, and likely cannot be modified or extended to take long-tailed distributions into account. In contrast, the learned heuristic, while not fully able to realize the total available advantage, performs much better. In both the agile and ultra-agile cases, the learned heuristic produces a schedule with approximately equal utility to that of the conventional schedule, indicating that the heuristic can appropriately estimate advantage and opportunity cost even in the case of Pareto-distributed task utility, especially given its much more limited use of lookaheads, less than half the lookaheads performed compare to the unit lookahead cases, indicating significantly higher estimated opportunity cost.

Overall, the Pareto-distributed case represents the most challenging case for dynamic tasking: increases in total utility are meager and there is tremendous downside risk. Even in this case, we have shown that dynamic tasking algorithms can improve or maintain schedule performance with additional awareness, implying that for any distribution of task utility from unit to extremely long-tailed, the DT algorithms presented in this work can still obtain schedule improvement.

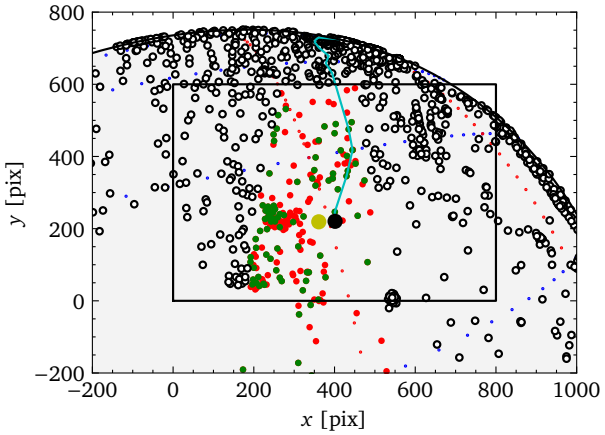
Table 6.7: Table of results for Pareto-distributed access utilities for agile satellite case, showing lookahead heuristic, initial and final schedule lengths $|\boldsymbol{\sigma}|$, initial and final schedule utilities $J_{\boldsymbol{\sigma}}$, number of lookaheads $N_{\text{lookaheads}}$, and improvement per lookahead $\Delta J / N_{\text{lookaheads}}$.

Case	**Heuristic**	$\lvert\boldsymbol{\sigma}_{\text{final}}\rvert$	$J_{\boldsymbol{\sigma}_{\text{final}}}$	$N_{\text{lookaheads}}$	$\Delta J / N_{\text{lookaheads}}$	$\widehat{\Delta J}$
Conventional		555.8 (541–564)	29907.1 (15295.6–42707.3)			
Omniscient		431.4 (422–440)	31216.7 (18578–43791.5)			
Lookahead	Stationary	442.3 (431–450)	30761.5 (18289.4–43459.3)			
Lookahead	Analytic	358.1 (349–365)	19858.3 (9014.2–37238.0)	115.3 (110–120)	-87.2 (-15739.4–2010.1)	1.72 (0.0–7.3)
Lookahead	Learned	486.5 (459–403)	29921.2 (14917.7–42657.4)	44.15 (33–50)	0.32 (-853.6–423.7)	15.95 (0.20–1142.55)
Meteorological ISL 30 m		425.8 (418.0–435.0)	29656.7 (17484.2–42233.3)			
Meteorological ISL 60 m		430.7 (423.0–439.0)	29224.4 (17103.0–42658.2)			
Meteorological ISL 90 m		436.4 (428.0–451.0)	28479.20 (15077.8–42581.6)			
Meteorological ISL 120 m		438.0 (432.0–447.0)	28552.0 (14953.2–42404.6)			
Meteorological G/S 0 m		423.0 (417–427)	29920.4 (17593.0–43230.1)			
Meteorological G/S 60 m		433.8 (423.0–442.0)	28746.8 (14881.6–42446.2)			
Meteorological G/S 30 m		432.8 (425–441)	29656.7 (17484.2–42233.3)			
Meteorological G/S 90 m		441.7 (434.0–453.0)	28485.4 (14938.5–42245.4)			
Meteorological G/S 120 m		438.0 (432.0–447.0)	28552.0 (14953.2–42404.6)			

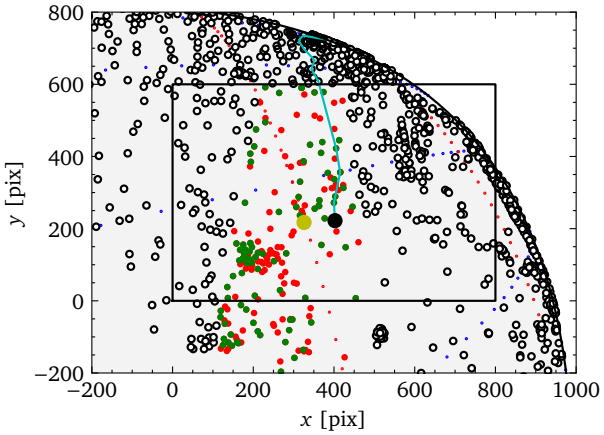
Table 6.8: Table of results for Pareto-distributed access utilities for ultra-agile satellite case, showing lookahead heuristic, initial and final schedule lengths $|\boldsymbol{\sigma}|$, initial and final schedule utilities $J_{\boldsymbol{\sigma}}$, number of lookaheads $N_{\text{lookaheads}}$, improvement per lookahead $\Delta J / N_{\text{lookaheads}}$, and estimated improvement per lookahead $\widehat{\Delta J}$.

Case	**Heuristic**	$\lvert\boldsymbol{\sigma}_{\text{final}}\rvert$	$J_{\boldsymbol{\sigma}_{\text{final}}}$	$N_{\text{lookaheads}}$	$\Delta J / N_{\text{lookaheads}}$	$\widehat{\Delta J}$
Conventional		878.4 (864–894)	32384.0 (19879–44232)			
Omniscient		646.7 (633–670)	33555.4 (20939.7–45961.5)			
Lookahead	Stationary	662.1 (649–680)	32399.5 (19914.8–45597.0)			
Lookahead	Analytic	479.3 (469–496)	21222.6 (12487.9–38667.0)	222.1 (216–226)	-50.5 (-13833.5–733.5)	1.42 (0.0–13.1)
Lookahead	Learned	752.6 (732–783)	32197.7 (19219.4–45025.0)	57.5 (44–69)	-3.25 (-3779.2–753.4)	15.75 (0.0–1278.0)
Meteorological ISL 30 m		638.45 (625.00–661.00)	31903.32 (19706.39–44430.52)			
Meteorological ISL 60 m		647.55 (630.00–665.00)	31409.25 (19317.98–44673.88)			
Meteorological ISL 90 m		655.00 (647.00–670.00)	30718.14 (19186.37–44712.58)			
Meteorological ISL 120 m		654.95 (643.00–674.00)	30639.27 (17131.16–43851.02)			
Meteorological G/S 0 m		636.05 (624.00–659.00)	32091.55 (19739.63–44597.72)			
Meteorological G/S 30 m		647.85 (636.00–667.00)	31504.86 (19455.78–44324.76)			
Meteorological G/S 60 m		653.30 (636.00–671.00)	30904.96 (19080.11–43960.96)			
Meteorological G/S 90 m		658.90 (644.00–673.00)	30570.54 (17077.61–43931.64)			
Meteorological G/S 120 m		660.25 (643.00–676.00)	30471.96 (16373.09–43439.06)			

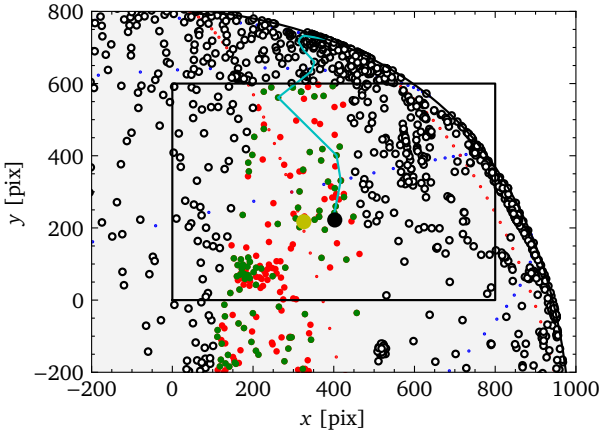
6.7 Camera Simulation



(a) Directly before imaging a task.



(b) New tasks about to enter lookahead ield of view.



(c) Schedule optimized based on new vision inputs.

Figure 6.23: Example simulation of stationary dynamic tasking case. Empty dots represent unobserved or unreachable accesses, green dots represent observed cloud-free accesses, and red dots represent cloudy accesses. The black box represents the lookahead sensor field of view, with the primary instrument track represented by the yellow dot, nadir track represented by the large black dot, and the current schedule is represented as the cyan line between accesses. As (a) the spacecraft is executing its schedule, (b) new accesses come into the field of view of lookahead sensor, get analyzed and (c) a new schedule is obtained, within a single overflight.

We present a camera simulation in Figure 6.23 to show how an example of the stationary vision-based DT case functions. In a single overflight, accesses enter the field of view of the lookahead sensor, are analyzed for their cloud state, and the schedule is re-optimized to maximize the number of cloud-free collects.

6.8 Summary

This chapter introduced the definition of dynamic tasking as an adaptive stochastic orienteering problem (ASOP) and developed practical algorithms for onboard implementation. Instead of solving the ASOP directly, we also introduce the heuristics used to extend previous scheduling work to dynamic tasking, and evaluate across various simulation environments. Evaluation across agile and ultra-agile spacecraft agility cases, body-fixed lookahead and meteorological data, and Pareto-distributed task utilities demonstrated that heuristic-driven dynamic tasking consistently improves or sustains useful image throughput over conventional scheduling, while remaining lightweight and viable to run on edge computing hardware.

Chapter 7

Constellation Extensions

The heuristics presented in §6 Sensing and Dynamic Tasking can be extended to constellations, under the assumption that we consider constellation-wide operations the same as that of individual operations, but with a global belief state. Provided that data about accesses that are shared by multiple spacecraft is allocated efficiently, and can be shared with some regular cadence, such as through an ISL, the same heuristics can work without further modification. A diagram depicting this global belief state is shown in Figure 7.2. We consider only task redistribution and awareness across a constellation with no modification of the constellation ephemerides, and with no focus on global optimality, rather the minimum that would be necessary to reproduce dynamic tasking across a constellation.

Dynamic tasking is likely to improve with a constellation with a global belief state, as additional data from lookaheads can be shared and used, provided that a channel such as an inter-satellite link (ISL) or a delayed networking approach such as through ground stations is used. To investigate the performance in the case of a constellation, we configure the agile with an additional follower spacecraft, resulting in a leader-follower constellation consisting of two identical spacecraft in the same orbit, separated by 30 minutes. We assume that there is an ISL available to transmit belief state information between the two spacecraft. A leader-follower configuration provides two important comparison cases: due to Earth's rotation, accesses at equatorial latitudes are mostly separate, while at northerly latitudes they become more shared, which are both important comparison cases for dynamic tasking. Depending on the separation in time, more or fewer accesses can be shared. For a leader-follower constellation separated by constant phase, the separation in time increases, leading to fewer shared accesses e.g. in 550 km SSO compared to 400 km LEO.

Figure 7.3 shows an example simulation of the considered leader-follower dynamic tasking scenario over Europe and into the Middle East. At higher latitudes, the follower spacecraft performs no lookaheads because the leader has already observed all visible tasks in that region. However, due to the effect of Earth's rotation causing the follower's orbital ground track to drift eastward over time, new, unobserved tasks cause the follower perform additional lookaheads. A set of unobserved accesses that looks like they have been covered by the follower's lookaheads are marked as unobserved, since they are unreachable by the follower

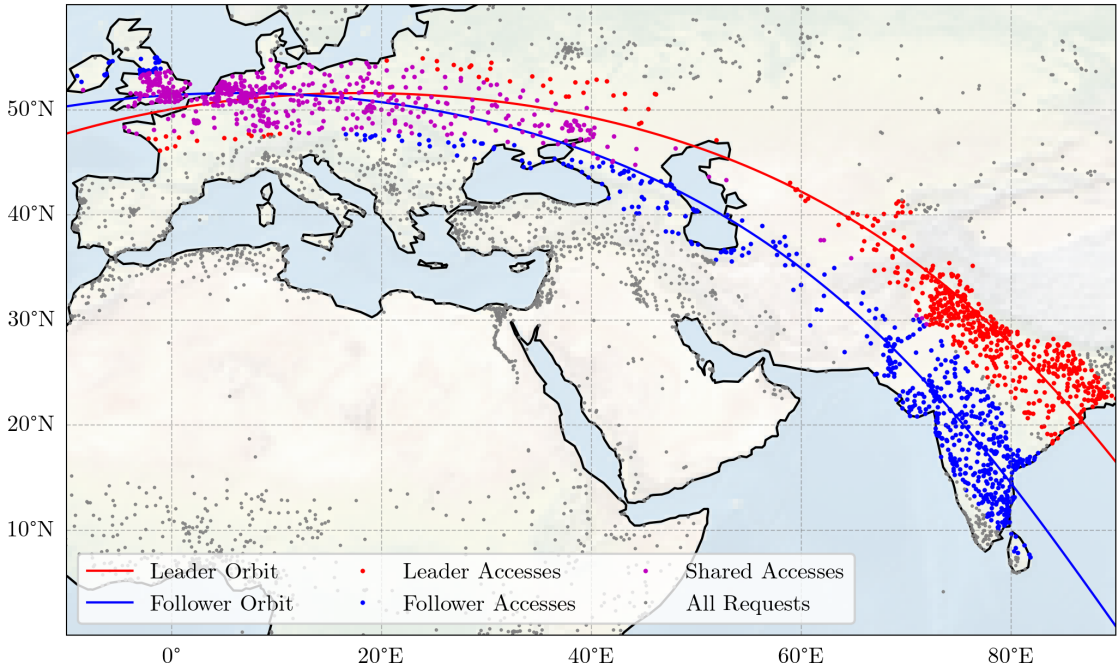


Figure 7.1: Orbit track and exclusive and shared requests for a leader-follower, showing how request sharing is more common at higher latitudes due to Earth's rotation. Leader and follower are both in 51.6° orbits with the follower trailing by half an orbit.

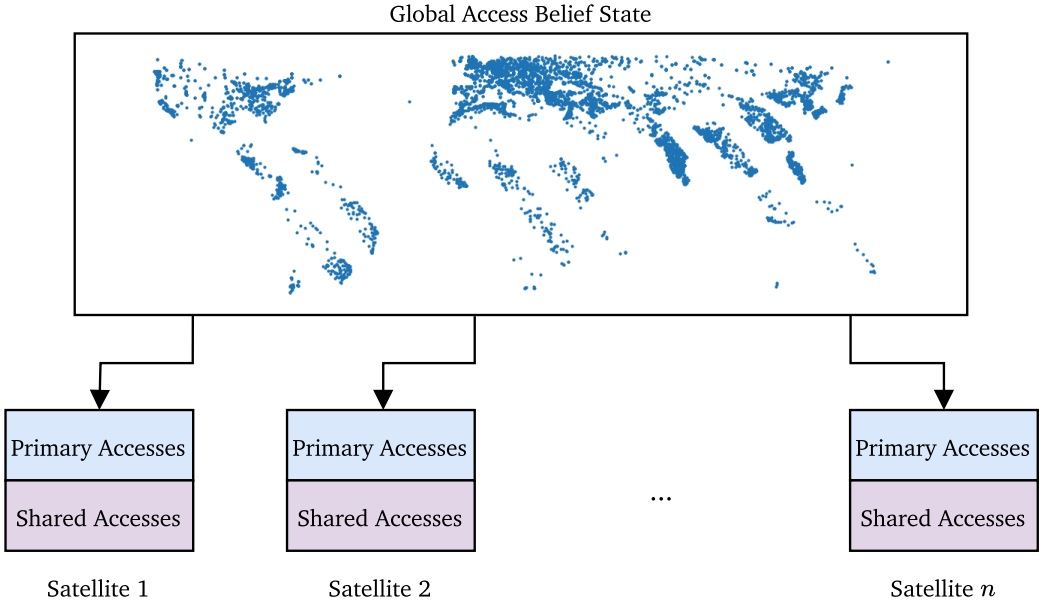


Figure 7.2: Diagram showing how a global belief state can be constructed from a distributed set of belief states, provided that shared requests are updated and maintained.

and the simulation follows causality (i.e. those accesses are in the past).

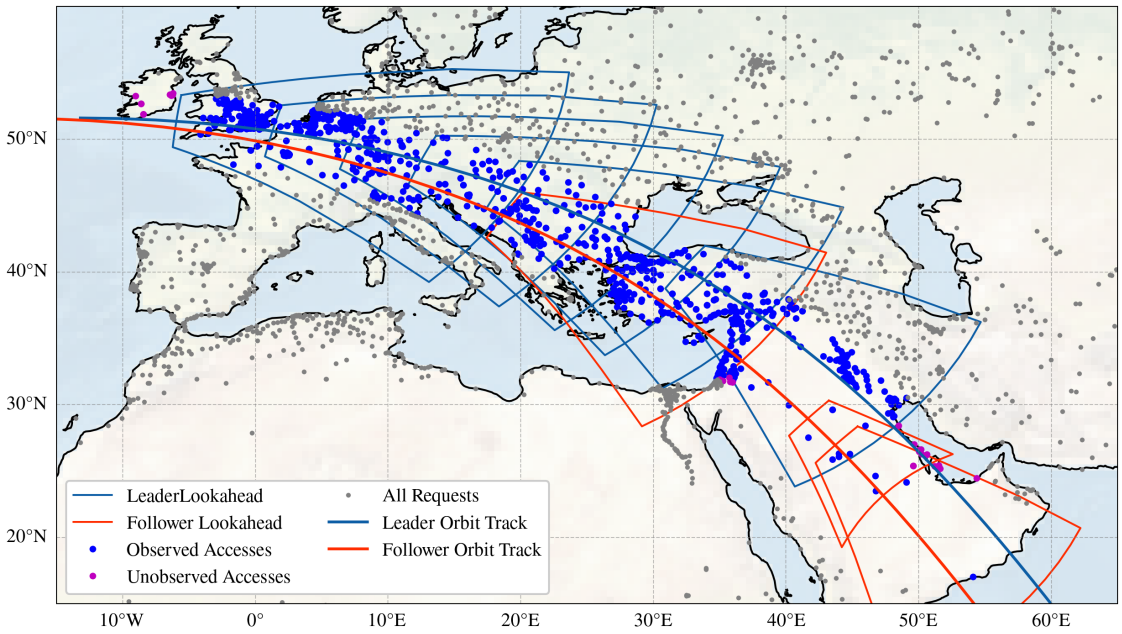


Figure 7.3: Example leader-follower dynamic tasking simulation run over Europe. Follower performs no lookaheads at northerly latitudes since the leader has already covered all the tasks, but due to Earth's rotation shifting the ground track, the follower performs more at mid-latitudes. Some tasks observed by the follower but in the ground track of the leader are marked unobserved since the simulation enforces causality.

While the data sharing strategy has been shown to work for the two-satellite constellation considered, allocating the data and keeping everything in synchronization can create additional nonlinear complexity with the size of constellation. In this case, the same result can be achieved trading computation requirements for data and bandwidth ones by duplicating and transmitting global belief state to every satellite. However, this type of allocation is likely only beneficial for extremely large constellations of approximately 1,000 satellites or more.

7.1 Summary

This chapter extended dynamic tasking heuristics from single spacecraft to constellations without focus on global optimality by introducing a shared global belief state. Simulating with a leader-follower constellation, results showed that inter-satellite information sharing enables task redistribution and reduces redundant lookaheads, with latitude-dependent effects from Earth's rotation affecting how many imaging requests are shared between the constellation.

Chapter 8

Conclusions & Future Work

Overall, this work shows that using vision instruments to assess upcoming image opportunities through agile lookahead is a viable mission architecture for agile Earth-observing satellite constellations equipped with onboard edge computing. In the case of unit task utility we find improvement in useful image throughput in all trialed cases, while when using more realistic Pareto-distributed utility, dynamic tasking can still find improvements to the schedule, despite the gap between the omniscient and conventional schedule significantly narrowing and schedule improvement becoming harder to find.

Future work includes many different avenues of exploration, ranging from technology demonstration to a variety of algorithmic improvements. This work assessing overall mission types is naturally highly parameterized and necessarily has had to make simplifications, as a single scheduling study cannot possibly encapsulate all the complexities that can exist in real missions. Significant optimization could be performed when analyzing a specific mission type instead, especially with continuous monitoring during operations.

There are many additional factors to consider when implementing for a specific mission, and many additional challenges to tackle. While this thesis tackles some of the theoretical and algorithmic frameworks, there are still significant complexities to overcome when implementing on a typical EO spacecraft. These complexities include additional resource requirements, slewing requirements such as rotational velocity constraints, instrument specifics such as warm-up times, imaging duty cycle, constraints on resolution, and so on.

While this thesis looks at continuous imaging, typical EO spacecraft image only a fraction of the orbit cycle, ranging typically between 5-20%, constrained primarily by energy available onboard the spacecraft. Usage of onboard computing and additional required slews affect the same power requirements, and an additional trade-off may need to be made between imaging and lookaheads from the power side.

Additionally, while this work assumes perfect state estimation from vision, in reality there is some accuracy attached to the vision system dependent on resolution. We may not also want binary classifications but rather probabilities or cloud thresholds, which can be incorporated into the utility function. Ground resolution constraints, if known, could also be potentially

mitigated by applying action limits to lookahead actions, e.g. not allowing any lookaheads pitching over 20°.

With the advent of new, compact hyperspectral sensors being flown on small satellites, while the spatial resolution may not yet meet the needs of extremely high resolution (e.g. 30 cm) imagery of tasked spacecraft, they could potentially be extremely useful as lookahead instruments due to the high specificity provided by the bands combined with spectral unmixing or spectral search algorithms [79].

There is a large gap in this work still between the dynamically tasked schedules and the omniscient schedule—additional sensor fusion, improved prediction, and more data likely can improve performance without the same opportunity costs that agile lookahead entails. Using meteorological data as a first pass [108, 109] and deriving a large dataset of conditional prediction probabilities could likely significantly improve the performance, getting much closer to the upper bound set by the omniscient case.

Now that this work has shown that the heuristics developed are capable of accurately estimating tradeoffs, there is likely significant benefit that can be obtained by adding additional global planning. In this work, all actions are greedy and performed only when the tradeoff is net-positive on a local level, which can potentially cause

The constellation extensions in this work also require an inter-satellite link to be present, so that belief states can be synchronized. While this technology is becoming more common, it is still rare and often costly to transmit even small messages. Delay-tolerant scheduling using statistical measures instead of the exact belief state and then synchronizing over ground station overpasses may be sufficient [94].

The post-processing approach to adding lookaheads in this work cuts down search space complexity significantly, but is not likely to achieve optimal solutions due to the inherently greedy nature of running these algorithms only on the spacecraft. More optimal solutions can likely be found by pre-planning lookaheads in the initial ground schedule, using an algorithm such as Monte Carlo Tree Search (MCTS) as used by AlphaGo [160] and previously applied to scheduling for agile Earth-observing spacecraft [132]. While this technique could potentially result in improvements in schedule performance and could implicitly encode a global planner into the policy, significant additional effort would have to go into operationalization due to the nature of the algorithm being much more of a black box with aspects of stochasticity.

This work focuses primarily on uses cases for commercial EO with optical imagery, however the concepts and analysis developed for this thesis are broader than the application considered. Future work could extend analysis to science-focused missions, with a variety of use cases for unpredictable phenomena both on Earth and potentially elsewhere. Science uses cases for dynamic targeting include but are certainly not limited to smart tracking of deep convective ice storm features [84] and planetary boundary layer tracking and sensing [51].

Appendix A

Core Implementation Details

This appendix covers the core implementation features, from which dynamic tasking simulations can be exactly constructed.

```python
# access.py
import numpy as np
import datetime
import itertools
from tqdm import tqdm

from collections import namedtuple
from dynamic_tasker.orbits import *
from dynamic_tasker.vector import *
from dynamic_tasker.spacecraft import *

class Request:
    def __init__(self, id, lat, long, name, utility=1):
        self.id = id
        self.lat = lat
        self.long = long
        self.name = name
        self.utility = utility

    def __repr__(self):
        return f"Request({self.id}, {self.lat}, {self.long}, {self.name}, {self.utility})"

class Access:
    def __init__(self, request, time, angle, state=None, utility=None):
        self.request = request
        self.requestid = request.id
        self.lat = self.request.lat
        self.long = self.request.long
        self.name = self.request.name
        self.time = time
        self.angle = angle
        self.state = state

        if(utility is not None):
            self.utility = utility
        else:
            self.utility = self.request.utility

    def __repr__(self):
        return f"Access({self.requestid}, {self.lat}, {self.long}, {self.name}, {self.time}, {self.angle},
        ↪  {self.state}, {self.utility})"

def multi_async_dispatch_search(requests, req_latlongs, field_of_regard, t0_s, t0, t1, t2, r1, r2, d1, d2, orbit):
```

```
    t3 = (t1 + t2) / 2
    # 1 second threshold
    if(t2 - t1 < 1):
        tasks = []
        for i, x in enumerate(requests):
            # Check access constraints...
            # Propagate to position
            task_eci = ecef2eci(latlong2ecef((x.lat, x.long)), t0 + datetime.timedelta(seconds=t3 - t0_s))
            r, v = kepler2eci(propagate_orbit(orbit, t1))
            # Compute the angle using dot product
            angle_diff = np.arccos(np.dot(r, r - task_eci) / (np.linalg.norm(r) * np.linalg.norm(task_eci - r)))
            sign = np.sign(dist2plane(r, np.cross(r, v), task_eci))
            angle_diff = np.rad2deg(angle_diff) * sign
            if(np.abs(angle_diff) < field_of_regard and task_not_in_eclipse(t0 + datetime.timedelta(seconds=t3 -
            ↪  t0_s), r)):
                tasks.append(Access(x, t0 + datetime.timedelta(seconds=t3 - t0_s), angle_diff))

        return tasks

    # Assume all tasks change sign in this time period
    r3, v3 = kepler2eci(propagate_orbit(orbit, t3))

    tasks_eci_3 = ecef2eci_vec(latlong2ecef_vec(req_latlongs), t0 + datetime.timedelta(seconds=t3 - t0_s))

    d3 = dist2plane(r3, v3, tasks_eci_3)

    filter_firsthalf = d1 * d3 < 0
    filter_secondhalf = d2 * d3 < 0

    tasks_firsthalf = [requests[i] for i in range(len(requests)) if filter_firsthalf[i]]
    tasks_secondhalf = [requests[i] for i in range(len(requests)) if filter_secondhalf[i]]

    task_times_1 = []
    task_times_2 = []

    # Split and recurse
    if(len(tasks_firsthalf) > 0):
        task_times_1 = multi_async_dispatch_search(tasks_firsthalf, req_latlongs[filter_firsthalf],
        ↪  field_of_regard, t0_s, t0, t1, t3, r1, r3, d1[filter_firsthalf], d3[filter_firsthalf], orbit)

    if(len(tasks_secondhalf) > 0):
        task_times_2 = multi_async_dispatch_search(tasks_secondhalf, req_latlongs[filter_secondhalf],
        ↪  field_of_regard, t0_s, t0, t3, t2, r3, r2, d3[filter_secondhalf], d2[filter_secondhalf], orbit)

    return task_times_1 + task_times_2

def get_accesses(requests, orbit, t_coarse, field_of_regard, t0, t_end):
    h = orbit.a - Constants.R_E # assume circular orbit
    v = v_orb(h)
    theta = np.arctan((h * np.tan(np.deg2rad(field_of_regard))) / Constants.R_E) # Max possible lateral angle
    theta_total = theta + np.deg2rad((Constants.gamma / (24 * 3600)) * t_coarse) # Worst case Earth rotation
    filter_radius = np.sqrt((v * t_coarse/2)**2 + (h + Constants.R_E * (1 - np.cos(theta_total)))**2 +
    ↪  (Constants.R_E * np.sin(theta_total))**2)

    seconds_since_epoch = 0 #(t0 - Constants.J2000).total_seconds()

    total_tasks = []
    req_latlongs = np.array([[x.lat, x.long] for x in requests])

    r1, v1 = kepler2eci(propagate_orbit(orbit, seconds_since_epoch))
    for i in range(0, int((t_end - t0).total_seconds()) - t_coarse, t_coarse):
        t1 = seconds_since_epoch + i
        t2 = seconds_since_epoch + i + t_coarse

        # Calculate ECI points
        r2, v2 = kepler2eci(propagate_orbit(orbit, t2))

        tasks_eci_1 = ecef2eci_vec(latlong2ecef_vec(req_latlongs), t0 + datetime.timedelta(seconds=i))
```

```
        tasks_eci_2 = ecef2eci_vec(latlong2ecef_vec(req_latlongs), t0 + datetime.timedelta(seconds=i + t_coarse))

        task_dists_1 = dist(r1, tasks_eci_1)
        task_dists_2 = dist(r2, tasks_eci_2)

        task_mask = (task_dists_1 <= filter_radius) + (task_dists_2 <= filter_radius)

        if(np.sum(task_mask) > 0):
            requests_prefiltered = list(itertools.compress(requests, task_mask))
            req_latlongs_prefiltered = list(itertools.compress(req_latlongs, task_mask))

            d1 = dist2plane(r1, v1, tasks_eci_1[task_mask])
            d2 = dist2plane(r2, v2, tasks_eci_2[task_mask])

            requests_crossindex = [d1[i] * d2[i] < 0 and d1[i] > d2[i] for i in range(len(requests_prefiltered))]

            if(np.sum(requests_crossindex) > 0):
                requests_filtered = list(itertools.compress(requests_prefiltered, requests_crossindex))
                req_latlongs_filtered = np.array(list(itertools.compress(req_latlongs_prefiltered,
                ↪  requests_crossindex)))
                new_tasks = multi_async_dispatch_search(requests_filtered, req_latlongs_filtered, field_of_regard,
                ↪  seconds_since_epoch, t0, t1, t2, r1, r2, d1[requests_crossindex], d2[requests_crossindex],
                ↪  orbit)
                total_tasks += new_tasks

        r1 = r2
        v1 = v2
    return total_tasks

def task_not_in_eclipse(time, r):
    sun_vector = sunvec_eci(time)
    return np.dot(r, sun_vector) > 0
```

```
# areas.py
import numpy as np
from itertools import combinations

# ---------- small utilities ------------------------------------------------
EPS = 1e-12

def normalize(v: np.ndarray) -> np.ndarray:
    """Return v / ‖v‖."""
    n = np.linalg.norm(v)
    if n < EPS:
        raise ValueError("zero-length vector")
    return v / n

def interior_signs(planes, interior_pt):
    """
    Return _j = ±1 for every plane, where +1 means the interior point is on
    the 'inside' side ( n·x > d ).
    """
    return [1 if np.dot(p["n"], interior_pt) - p["d"] > 0 else -1
            for p in planes]


def vertex_angle(v, n_prev, n_next):
    """
    Corner angle  at vertex v where the boundary turns from plane n_prev
    to plane n_next (both unit normals).
    """
    t1 = normalize(np.cross(n_prev, v))
    t2 = normalize(np.cross(n_next, v))
    return np.arccos(np.clip(np.dot(t1, t2), -1.0, 1.0))


def edge_contribution(v1, v2, n, d, sigma):
    """
```

```
     _g ds of the small-circle arc between v1 and v2 on plane (n,d),
    with orientation sign sigma = ±1.
    """
     = np.arccos(d)                 # angular radius of the circle
    if abs(np.cos()) < 1e-14:       # great circle  _g = 0
        return 0.0

    # project the vertices onto the plane to obtain circle centre-angle
    u1 = v1 - d * n
    u2 = v2 - d * n
    denom = np.sin() ** 2
    cos_ = np.clip(np.dot(u1, u2) / denom, -1.0, 1.0)
     = np.arccos(cos_)

    return sigma * np.cos() *
# ----------------------------------------------------------------------------

def already_used(u, used):
    return any(np.allclose(u, w, atol=EPS) for w in used)

def spherical_polygon_area(vertices,                    # [N,3]
                           planes,                      # list of {"n":..., "d":...}
                           edge_plane_indices,          # [N] plane index for edge i→i+1
                           interior_pt=None):
    """
    Area of the region on the unit sphere bounded by the given planes.

    Parameters
    ----------
    vertices            CCW-ordered vertices on S² (N×3 array-like)
    planes              [{'n': unit normal (3,), 'd': scalar}, . . . ]
    edge_plane_indices  plane index of edge (v_i → v_{i+1})
    interior_pt         optional point known to lie inside the patch
                        (defaults to mean of vertices, then renormalised)

    Returns
    -------
    area  (float) - steradians
    """
    V = [normalize(np.asarray(v, float)) for v in vertices]
    N = len(V)

    if interior_pt is None:
        interior_pt = normalize(np.mean(V, axis=0))

     = interior_signs(planes, interior_pt)

    corner_sum = 0.0
    edge_sum   = 0.0

    for i in range(N):
        v_curr = V[i]
        v_next = V[(i + 1) % N]

        plane_prev = planes[edge_plane_indices[i - 1]]
        plane_next = planes[edge_plane_indices[i]]

        # --- corner term _i ------------------------------------------------
        corner_sum += vertex_angle(v_curr,
                                   plane_prev["n"],
                                   plane_next["n"])

        # --- edge line-integral term ---------------------------------------
        idx = edge_plane_indices[i]
        p   = planes[idx]
        edge_sum += edge_contribution(v_curr, v_next,
                                      p["n"], p["d"], [idx])

```

```
    # Gauss-Bonnet on S²  →  A = 2  _i  _g ds
    return 2.0 * np.pi - corner_sum - edge_sum

def are_planes_equal(p1, p2, tol=1e-9):
    return (np.allclose(p1["n"], p2["n"], atol=tol) and
            np.isclose(p1["d"],  p2["d"], atol=tol))


def vertex_set_from_planes(planes, interior_pt=None):
    # Start first by calculating all intersections of the planes
    lines = []

    # Find the intersection of each pair of planes
    for p,q in combinations(planes, 2):
        # if np.all(p["n"] == q["n"]) and np.all(p["d"] == q["d"]):
        #     continue

        n1 = p["n"]
        n2 = q["n"]
        d1 = p["d"]
        d2 = q["d"]

        v = np.cross(n1, n2)
        if np.linalg.norm(v) < EPS:
            continue

        p0 = np.cross(d1 * n2 - d2 * n1, v)/np.linalg.norm(v)**2

        lines.append((p0, v))

    vertices = []
    # Calculate vertices as lines projected to unit magnitude
    for i, (p0, v) in enumerate(lines):
        a = v.dot(v)
        b = 2 * p0.dot(v)
        c = p0.dot(p0) - 1.0

        delta = b**2 - 4 * a * c
        if delta < 0:
            continue

        sqrt_delta = np.sqrt(delta)

        t1 = (-b + sqrt_delta) / (2 * a)
        t2 = (-b - sqrt_delta) / (2 * a)

        vertices.append(p0 + t1 * v)
        vertices.append(p0 + t2 * v)

    # Renormalize the vertices to unit length
    vertices = [normalize(v) for v in vertices]

    # Remove ones that aren't tightly bound by the planes
    vertices_tight = [v for v in vertices if np.all([np.dot(p["n"], v) - p["d"] >= -EPS for p in planes])]
    # Order the vertices
    vertices_sorted = [vertices_tight[0]]
    planes_used = []
    edge_plane_indices = []


    # centre = normalize(np.mean(vertices_tight, axis=0))
    # choose an arbitrary in-plane x-axis
    centre = normalize(np.mean(vertices_tight, axis=0)) if interior_pt is None else normalize(interior_pt)
    x0 = (np.cross([1, 0, 0], centre))
    if np.linalg.norm(x0) < EPS:               # centre  pole
        x0 = np.array([1, 0, 0])
    else:
        x0 = normalize(x0)
```

```
    y0 = np.cross(centre, x0)

    angles = [np.arctan2(np.dot(y0, v), np.dot(x0, v)) for v in vertices_tight]
    order  = np.argsort(angles)
    vertices_sorted = [vertices_tight[i] for i in order]

    # for i in range(len(vertices_tight) - 1):
    #     v = vertices_sorted[-1]
    #     # Find the plane that is closest to the vertex
    #     planes_close = [p for p in planes if np.abs(np.dot(p["n"], v) - p["d"]) <= EPS and (True if
    ↪  len(planes_used) == 0 else not are_planes_equal(p, planes_used[-1]))]
    #     if len(planes_close) == 0:
    #         raise ValueError("No plane found for vertex")
    #     plane = planes_close[-1]
    #     planes_used.append(plane)
    #     # edge_plane_indices.append(planes.index(plane))
    #     edge_plane_indices.append([i for i, p in enumerate(planes) if are_planes_equal(p, plane)][0])

    #     # Find the next vertex that is closest to the plane
    #     next_vertex = None
    #     next_vertex = [v for v in vertices_tight if np.abs(np.dot(plane["n"], v) - plane["d"]) <= EPS and (v !=
    ↪  vertices_sorted[-1]).any()]
    #     if next_vertex:
    #         vertices_sorted.append(next_vertex[0])
    #     else:
    #         raise ValueError("No next vertex found for the current vertex")

    # # Add the final plane to the list
    # # This plane connects the last vertex to the first vertex
    # plane_final = [p for p in planes if np.abs(np.dot(p["n"], vertices_sorted[-1]) - p["d"]) <= EPS and
    ↪  np.abs(np.dot(p["n"], vertices_sorted[0]) - p["d"]) <= EPS]
    # if len(plane_final) == 0:
    #     raise ValueError("No final plane found for the last vertex")

    # planes_used.append(plane_final[-1])
    # edge_plane_indices.append([i for i, p in enumerate(planes) if are_planes_equal(p, plane_final[-1])][0])
    edge_plane_indices = []
    for i in range(len(vertices_sorted)):
        v_cur, v_nxt = vertices_sorted[i], vertices_sorted[(i+1) % len(vertices_sorted)]
        shared = [k for k,p in enumerate(planes)
                if abs(np.dot(p["n"], v_cur) - p["d"]) <= EPS
                and abs(np.dot(p["n"], v_nxt) - p["d"]) <= EPS]

        # pick the plane that is *not* the previous one to maintain CCW order
        if i and shared[0] == edge_plane_indices[-1]:
            edge_plane_indices.append(shared[1])
        else:
            edge_plane_indices.append(shared[0])

    return vertices_sorted, planes, edge_plane_indices
```

```
# cameras.py
import numpy as np
from dynamic_tasker.orbits import *
from dynamic_tasker.rotations import *
from dynamic_tasker.vector import *
# Origin at bottom left corner
def get_intrinsics(f, c_x, c_y):
    K = np.hstack([np.array([-f, 0, c_x, 0, f, c_y, 0, 0, 1]).reshape(3,3), np.zeros((3,1))])
    return K

def get_intrinsics_from_fov(fov, width, height, axis='x'):
    if axis == 'x':
        f = width / (2 * np.tan(np.deg2rad(fov / 2)))
    elif axis == 'y':
        f = height / (2 * np.tan(np.deg2rad(fov / 2)))
    else:
```

```
        raise ValueError(f"Invalid axis: {axis}")

    return get_intrinsics(f, width/2, height/2)

def get_extrinsics(R_t, p):
    R_q = R_t #q2mat(q)
    R_0 = np.eye(3) # eul2R(-np.pi/4, np.pi/2, 0) # boresight on -z
    R = R_q @ R_0
    t = -R @ p
    bottom_row = np.array([[0, 0, 0, 1]])
    Rt = np.hstack([R, t.reshape(3, 1)])
    transformation_matrix = np.vstack([Rt, bottom_row])
    return transformation_matrix

def get_camera_matrix(K, R, p):
    return K @ get_extrinsics(R, p)

def project(P, points, z_clip=True):
    # Project points to image plane
    points = np.concatenate((points, np.ones((points.shape[0], 1))), axis=1)
    points = (P @ points.T).T

    # Clip all points with z < 0
    if(z_clip):
        points = points[points[:, 2] > 0]

    points = points / points[:, [2]]
    points = points[:, 0:2]

    return points

def unproject(P, img_points, depth):
    """
    Unprojects 2D image points to 3D world coordinates given their depth values.

    Parameters
    ----------
    P : np.ndarray
        The 3x4 camera projection matrix (assumed to be of the form K[R | -R*t]).
    img_points : np.ndarray
        An (N x 2) array of image (pixel) coordinates.
    depth : float or np.ndarray
        The depth value(s) corresponding to the camera z-coordinate (before homogeneous division).
        This can be a single scalar (applied to all points) or an (N,) array.

    Returns
    -------
    world_points : np.ndarray
        An (N x 3) array of unprojected 3D world coordinates.

    Notes
    -----
    Because the projection operation loses the depth information, unprojection is ambiguous
    unless the depth is provided. This function inverts the operation:

        [u, v]  -->  d*[u, v, 1]^T = M*X + p4  with M = P[:, :3] and p4 = P[:, 3],

    and returns:

        X = inv(M) * (d*[u, v, 1]^T - p4)
    """
    # Ensure depth is an (N, 1) array.
    if np.isscalar(depth):
        depth = np.full((img_points.shape[0], 1), depth)
    else:
        depth = np.array(depth).reshape(-1, 1)

    # Decompose P into its 3x3 part and the translation part.
```

```
    M = P[:, :3]    # should be invertible (e.g. K*R)
    p4 = P[:, 3]    # shape (3,)

    # Build the homogeneous image coordinates scaled by the depth.
    # For each 2D point [u, v], we form [u*d, v*d, d]
    homogeneous_img = np.hstack([img_points * depth, depth])

    # Solve for the world point: X = inv(M) @ (d*[u,v,1] - p4)
    invM = np.linalg.inv(M)
    # Subtract p4 from each row (broadcasting works because p4 is (3,))
    world_points = (homogeneous_img - p4) @ invM.T

    return world_points

def project_from_orbit(points, K, orbit, time, roll_angle=0, pitch_angle=15):
    r, v = kepler2eci(propagate_orbit(orbit, time))
    # Figure out the quaternion pointing to nadir
    v_unit = v / np.linalg.norm(v)
    r_unit = r / np.linalg.norm(r)

    R_t = rotmat_from_vec(r_unit, -v_unit)
    R_q = eul2R(0, np.deg2rad(roll_angle), np.deg2rad(pitch_angle))
    R_t = R_t @ R_q
    R_t = np.linalg.inv(R_t)
    R_t = R_t[[2, 1, 0], :]
    P = get_camera_matrix(K, R_t, r)

    # Check if points is not empty
    if(points.size > 0):
        projected_points = project(P, points, False)
        return projected_points
    else:
        return []

def unproject_from_orbit(img_points, depth, K, orbit, time, roll_angle=0, pitch_angle=15):
    """
    Unprojects 2D image points to 3D world coordinates using orbit information.

    Parameters
    ----------
    img_points : np.ndarray
        An (N x 2) array of image (pixel) coordinates.
    depth : float or np.ndarray
        The depth value(s) corresponding to the camera z-coordinate (before homogeneous division).
        Can be a scalar (applied to all points) or an array of shape (N,).
    K : np.ndarray
        The 3x3 intrinsic calibration matrix.
    orbit : object
        The orbit parameters (or object) used by propagate_orbit and kepler2eci.
    time : float
        The time at which to propagate the orbit.
    roll_angle : float, optional
        The camera roll angle in degrees (default is 0).
    pitch_angle : float, optional
        The camera pitch angle in degrees (default is 15).

    Returns
    -------
    world_points : np.ndarray
        An (N x 3) array of 3D world coordinates corresponding to the unprojected image points.

    Notes
    -----
    This function first computes the camera position and orientation from the orbit parameters:

        - Propagate the orbit to the given time and convert to ECI coordinates.
        - Compute a rotation matrix pointing toward nadir using the unit vectors of the
          position (r_unit) and velocity (v_unit).
```

```
        - Adjust the rotation using the specified roll and pitch.
        - Compute the camera projection matrix P = get_camera_matrix(K, R_t, r).

    Then it uses the provided depth to unproject the 2D image points back into 3D space via:

        X = inv(M) @ (d*[u, v, 1]^T - p4)

    where M = P[:, :3] and p4 = P[:, 3].

    Dependencies
    ------------
    This function assumes that the following functions are defined elsewhere:
        - propagate_orbit(orbit, time)
        - kepler2eci(state)
        - rotmat_from_vec(target_vec, up_vec)
        - eul2R(yaw, pitch, roll)
        - get_camera_matrix(K, R, t)
        - unproject(P, img_points, depth)
    """
    # Propagate the orbit to the given time and convert to ECI coordinates.
    r, v = kepler2eci(propagate_orbit(orbit, time))

    # Compute unit vectors from position and velocity.
    r_unit = r / np.linalg.norm(r)
    v_unit = v / np.linalg.norm(v)

    # Compute the rotation matrix that aligns the camera with nadir (pointing toward Earth).
    R_t = rotmat_from_vec(r_unit, -v_unit)
    # Adjust for roll and pitch (convert angles from degrees to radians).
    R_q = eul2R(0, np.deg2rad(roll_angle), np.deg2rad(pitch_angle))
    R_t = R_t @ R_q
    # Invert the rotation to get the camera-to-world rotation.
    R_t = np.linalg.inv(R_t)
    # Reorder axes as in the project_from_orbit function.
    R_t = R_t[[2, 1, 0], :]

    # Construct the full camera projection matrix.
    P = get_camera_matrix(K, R_t, r)

    # Unproject the image points using the provided depth.
    world_points = unproject(P, img_points, depth)

    return world_points

def ecef2pitchroll(pos_ecef, v_ecef, vec):
    Up = pos_ecef / np.linalg.norm(pos_ecef)

    Along = v_ecef / np.linalg.norm(v_ecef)
    Right = np.cross(Along, Up)
    Right = Right / np.linalg.norm(Right)

    Along = np.cross(Up, Right)
    Along = Along / np.linalg.norm(Along)

    R_ecef_to_body = np.vstack([Right, Along, Up])

    v_local = R_ecef_to_body @ vec
    v_local_norm = v_local / np.linalg.norm(v_local)  # normalize for angle calculations

    pitch = -np.arctan2(v_local_norm[1], v_local_norm[2])
    roll  = -np.arctan2(v_local_norm[0], v_local_norm[2])

    pitch_deg = np.degrees(pitch) * 1
    roll_deg  = np.degrees(roll)

    return pitch_deg, roll_deg

def project_in_box(pitch_deg, roll_deg, orbit, t, accesses, points, width, height, K):
```

```
    # First, project and see if it's in the box
    points_eci = np.array([ecef2eci(p, t) for p in points])
    ecef_projected_dir = project_from_orbit(points_eci, K, orbit, t, pitch_angle=pitch_deg, roll_angle=roll_deg)
    # Figure out how many are in the box
    in_box_idx = np.array([i for i, p in enumerate(ecef_projected_dir) if p[0] >= 0 and p[0] <= width and p[1] >= 0 and p[1] <= height])
    return [a for i, a in enumerate(accesses) if i in in_box_idx], in_box_idx, ecef_projected_dir

def filter_accesses_horizon(orbit, time, accesses, pos_ecef, field_of_regard=30):
    return [(r, a, t, access, idx) for r, a, t, access, idx in accesses if t >= time and t <= time + datetime.timedelta(seconds=horizon_time(orbit)) and a <= field_of_regard and a >= -field_of_regard]

def create_box(width, height, points_per_edge=0):
    # Create the base box corners
    corners = np.array([[0, 0], [width, 0], [width, height], [0, height]])

    # Create high resolution edges
    edges = []
    for i in range(4):
        start = corners[i]
        end = corners[(i + 1) % 4]
        # Generate evenly spaced points along each edge
        edge_points = np.linspace(start, end, points_per_edge)
        edges.append(edge_points[:-1])  # Exclude last point to avoid duplicates

    # Combine all edges and add the closing point
    high_res_box = np.vstack(edges + [corners[0]])
    return high_res_box
```

```
# constants.py
import datetime

class Constants:
    # Time
    J2000 = datetime.datetime(2000, 1, 1, 12, 0, 0)
    seconds_per_day = 24 * 60 * 60


    # Earth
    R_E = 6378 # km
    mu = 398600.4418 # km^3/s^2
    ERA_J2000 = 280.46 # Earth rotation at J2000
    gamma=360.9856123035484 # Earth rotation rate, deg/day

    # Sun
```

```
# imagery.py
from collections import namedtuple
import numpy as np
import matplotlib as pyplot
import requests
import datetime
import os
import zipfile
import xml.etree.ElementTree as ET
from requests.auth import HTTPBasicAuth
import xarray as xr
# Linux only....
try:
    import pygrib
except ImportError:
    pass
import re
import pyproj
from pathlib import Path

dateformat = "%Y-%m-%d_%H%M%S"
data_dir = Path(__file__).resolve().parent.parent.parent / "data"
```

```python

image_sources = {
    "goes_west": {
        "url": "https://noaa-goes18.s3.amazonaws.com/",
        "description": "GOES West (18)",
        "product": "ABI-L2-ACMF",
        "lon": -137.2,
    },
    "goes_east": {
        "url": "https://noaa-goes16.s3.amazonaws.com/",
        "description": "GOES East (16)",
        "product": "ABI-L2-ACMF",
        "lon": -75.2,
    },
    "meteosat_zds": {
        "url": "https://api.eumetsat.int/data/download/1.0.0",
        "description": "Meteosat Zero Degree Service (ZDS)",
        "product": "EO:EUM:DAT:MSG:CLM",
        "lon": 0,
    },
    "meteosat_iodc": {
        "url": "https://api.eumetsat.int/data/download/1.0.0",
        "description": "Meteosat Indian Ocean Data Coverage (IODC)",
        "product": "EO:EUM:DAT:MSG:CLM-IODC",
        "lon": 45.5,
    },
    "himawari": {
        "url": "https://noaa-himawari9.s3.amazonaws.com/",
        "description": "Himawari 9",
        "product": "AHI-L2-FLDK-Clouds",
        "lon": 140.7,
    },
}

def download_goes_image(time: datetime, url, product, savefile, causal=True):
    if(time.minute == 0):
        time = time - datetime.timedelta(hours=1)

    year = time.year
    hour = time.hour
    minute = time.minute


    # Calculate day of year
    day_of_year = time.timetuple().tm_yday

    response = requests.get(f"{url}/?prefix={product}/{year}/{day_of_year:03d}/{hour:02d}/", verify=False)
    response.raise_for_status()

    # Parse xml tree
    root = ET.fromstring(response.text)

    # Find the match
    namespace = {"s3": "http://s3.amazonaws.com/doc/2006-03-01/"}
    filenames = [content.find("s3:Key", namespace).text for content in root.findall("s3:Contents", namespace)]
    # filename = [x for x in filenames if re.search(f"{minute:02d}[0-9]{{3}}\.nc", x)][0]
    filename = filenames[minute//10 if minute > 0 else -1]

    # Download the file
    response = requests.get(f"{url}/{filename}", verify=False)
    response.raise_for_status()

    # Save the file
    with open(os.path.join(data_dir, f"products/{savefile}.nc"), "wb") as f:
        f.write(response.content)

def download_goes_east_image(time: datetime):
```

```
    download_goes_image(time, image_sources["goes_east"]["url"], image_sources["goes_east"]["product"],
    ↪  f"goes_east_{time.strftime(dateformat)}")

def download_goes_west_image(time: datetime):
    download_goes_image(time, image_sources["goes_west"]["url"], image_sources["goes_west"]["product"],
    ↪  f"goes_west_{time.strftime(dateformat)}")

def download_meteosat_zds_image(time: datetime):
    token = get_auth_token_meteosat()
    download_meteosat_image(time=time, url=image_sources["meteosat_zds"]["url"], auth_token=token)

def download_meteosat_iodc_image(time: datetime):
    token = get_auth_token_meteosat()
    download_meteosat_image(time=time, url=image_sources["meteosat_iodc"]["url"], product="iodc",
    ↪  auth_token=token)

def download_himawari_image(time: datetime, url=image_sources["himawari"]["url"],
↪  product=image_sources["himawari"]["product"], savefile=None):
    year = time.year
    month = time.month
    day = time.day
    hour = time.hour
    minute = time.minute

    if(savefile is None):
        savefile = f"himawari_{time.strftime(dateformat)}"

    response = requests.get(f"{url}/?prefix={product}/{year}/{month:02d}/{day:02d}/{hour:02d}{minute:02d}",
    ↪  verify=False)
    response.raise_for_status()

    # Parse xml tree
    root = ET.fromstring(response.text)

    namespace = {"s3": "http://s3.amazonaws.com/doc/2006-03-01/"}
    filenames = [content.find("s3:Key", namespace).text for content in root.findall("s3:Contents", namespace)]

    filename = [x for x in filenames if "AHI-CMSK" in x][0]

    # Download the file
    response = requests.get(f"{url}/{filename}", verify=False)
    response.raise_for_status()

    # Save the file
    with open(os.path.join(data_dir, f"products/{savefile}.nc"), "wb") as f:
        f.write(response.content)

MeteosatAuthToken = namedtuple("MeteosatAuthToken", ["token", "expires"])

def get_auth_token_meteosat():
    # Get the auth token for the meteosat endpoint
    url = "https://api.eumetsat.int/token"
    # TODO: put keys in here!
    key = ""
    secret = ""

    data = {
        "grant_type": "client_credentials"
    }

    # Make the request
    response = requests.post(url, data=data, auth=HTTPBasicAuth(key, secret), verify=False)

    if(response.status_code == 200):
        json_response =  response.json()
        token = json_response["access_token"]
        expires = datetime.datetime.now() + datetime.timedelta(seconds=json_response["expires_in"])
        return MeteosatAuthToken(token, expires)
```

```
    else:
        raise Exception("Failed to get auth token for meteosat")

def revoke_auth_token_meteosat(auth_token: MeteosatAuthToken):
    url = "https://api.eumetsat.int/revoke"
    data = {
        "token": auth_token.token
    }

    response = requests.post(url, data=data, verify=False)

    if(response.status_code == 200):
        return True
    else:
        return False


def list_meteosat_images(auth_token: MeteosatAuthToken):
    # List the available images from the meteosat endpoint
    # url =
    pass

def download_meteosat_image(url, auth_token: MeteosatAuthToken, time, product="zds",
↪  tmp_folder=os.path.join(data_dir, "tmp"), products_folder=os.path.join(data_dir, "products"), causal=True):
    collection = "EO:EUM:DAT:MSG:CLM"

    if(product=="iodc"):
        collection += "-IODC"

    # Replace with web-safe product name
    collection = collection.replace(":", "%3A")
    year = time.year
    month = time.month
    day = time.day
    hour = time.hour
    minute = time.minute

    # round minute to nearest 15
    # if(causal):
    #     minute = int(np.floor(minute/15.0)) * 15
    # else:
    #     minute = int(np.round(minute/15.0)) * 15

    url = image_sources["meteosat_zds"]["url"]

    url += f"/collections/{collection}/dates/{year}/{month:02d}/{day:02d}/times/{hour:02d}/{minute:02d}?access_
    ↪  token={auth_token.token}"
    response = requests.get(url, headers={"Authorization": f"Bearer {auth_token.token}"}, verify=False)
    if(response.status_code != 200):
        raise Exception(f"Failed to download image: {response.status_code}, {response.text}")

    product_filename = os.path.join(tmp_folder, f"meteosat_{product}_{time.strftime(dateformat)}.zip")

    with open(product_filename, "wb") as f:
        f.write(response.content)

    # Unzip and save to data folder...
    with zipfile.ZipFile(product_filename, "r") as z:
        z.extractall(tmp_folder)

    # Now save the product .grb to the data folder...
    file_start = f"MSG{2 if product=='iodc' else
    ↪  3}-SEVI-MSGCLMK-0100-0100-{year}{month:02d}{day:02d}{hour:02d}{minute:02d}"
    # Search for the grb file
    for root, dirs, files in os.walk(tmp_folder):
        for file in files:
            if file.startswith(file_start) and file.endswith(".grb"):
```

```
                os.rename(os.path.join(root, file), os.path.join(products_folder,
                ↪  f"meteosat_{product}_{time.strftime(dateformat)}.grb"))
                break

image_sources = {
    "goes_west": {
        "url": "https://noaa-goes18.s3.amazonaws.com/",
        "description": "GOES West (18)",
        "product": "ABI-L2-ACMF",
        "lon": -137.2,
        "download": download_goes_west_image,
    },
    "goes_east": {
        "url": "https://noaa-goes16.s3.amazonaws.com/",
        "description": "GOES East (16)",
        "product": "ABI-L2-ACMF",
        "lon": -75.2,
        "download": download_goes_east_image,
    },
    "meteosat_zds": {
        "url": "https://api.eumetsat.int/data/download/1.0.0",
        "description": "Meteosat Zero Degree Service (ZDS)",
        "product": "EO:EUM:DAT:MSG:CLM",
        "lon": 0,
        "download": download_meteosat_zds_image,
    },
    "meteosat_iodc": {
        "url": "https://api.eumetsat.int/data/download/1.0.0",
        "description": "Meteosat Indian Ocean Data Coverage (IODC)",
        "product": "EO:EUM:DAT:MSG:CLM-IODC",
        "lon": 45.5,
        "download": download_meteosat_iodc_image,
    },
    "himawari": {
        "url": "https://noaa-himawari9.s3.amazonaws.com/",
        "description": "Himawari 9",
        "product": "AHI-L2-FLDK-Clouds",
        "lon": 140.7,
        "download": download_himawari_image,
    },
}

def get_closest_latlong_sample(data, lats, longs, point):
    lat, lon = point

    lats_1d = lats[:, lats.shape[0]//2]
    longs_1d = longs[longs.shape[1]//2, :]

    lat_idx = np.nanargmin(np.abs(lats_1d - lat))

    long_idx = np.nanargmin(np.abs(longs[lat_idx, :] - lon))

    # import matplotlib.pyplot as plt
    # plt.plot(np.abs(longs[lat_idx, :] - lon))
    # plt.show()

    return data[lat_idx, long_idx]

def extract_latlong_from_meteosat_grib(filename):
    grbs = pygrib.open(filename)
    cloud_mask = grbs[1].values
    lats = grbs[1].latitudes.reshape((3712, 3712))
    lons = grbs[1].longitudes.reshape((3712, 3712))

# def get_global_cloud_map(time: datetime):
#     # Get the global cloud map from the meteosat endpoint
#     auth_token = get_auth_token_meteosat()
#     download_meteosat_image(image_sources["meteosat_zds"]["url"], auth_token, time)
```

```python

def load_bcm(filename, lazy_load=False):
    # Load the binary cloud mask file
    # Get pure filename
    pure_filename = os.path.basename(filename)
    full_filename = os.path.join(data_dir, "products", filename)
    if(lazy_load and not os.path.exists(full_filename)):
        # Extract the date from the filename
        date = datetime.datetime.strptime("_".join(pure_filename.split(".")[0].split("_")[-2:]), dateformat)
        image_source = "_".join(pure_filename.split("_")[:-2])

        # Download the image
        image_sources[image_source]["download"](date)
        # Fall through to loading step

    if(pure_filename.startswith("goes")):
        with xr.open_dataset(full_filename) as ds:
            data = ds['BCM']

            proj_info = ds['goes_imager_projection'].attrs
            h = proj_info['perspective_point_height']
            lon_origin = proj_info['longitude_of_projection_origin']
            sweep_angle_axis = proj_info['sweep_angle_axis']
            # Define the projection
            proj = pyproj.Proj(proj='geos', h=h, lon_0=lon_origin, sweep=sweep_angle_axis)

            # Get x and y coordinate arrays
            x = ds['x'].values * h  # Scaled by perspective_point_height
            y = ds['y'].values * h

        # Create a meshgrid of x, y
        x_mesh, y_mesh = np.meshgrid(x, y)

        # Compute lat/lon from x/y
        lon, lat = proj(x_mesh, y_mesh, inverse=True)

        return data.to_numpy(), lat, lon

    elif(pure_filename.startswith("meteosat")):
        with pygrib.open(full_filename) as grbs:
            data = grbs[1].values
            lat = grbs[1].latitudes.reshape((3712, 3712))
            lon = grbs[1].longitudes.reshape((3712, 3712))

        lat[data == 3] = np.nan
        lon[data == 3] = np.nan

        lon[lon > 180] -= 360

        data[data == 3] = np.nan
        data[~np.isnan(data)] = (data[~np.isnan(data)] == 2)

        return data[::-1, ::-1], lat, lon

    elif(pure_filename.startswith("himawari")):
        with xr.open_dataset(full_filename) as ds:
            # Load the data
            data = ds['CloudMaskBinary'].values
            lat = ds["Latitude"].values
            lon = ds["Longitude"].values

        return data, lat, lon
    else:
        raise Exception("Unknown file type")

def sample_global_bcm(all_data, points):
    center_lons = [x["lon"] for x in image_sources.values()]
    center_lon_sorted = np.sort(center_lons)
```

```
    center_lon_cutoffs = (center_lon_sorted + np.roll(center_lon_sorted, 1)) / 2
    center_lon_cutoffs[0] -= 180

    # Create the points
    mask = np.zeros(len(points))
    for i, p in enumerate(points):
        lat, lon = p

        sat_idx = np.searchsorted(center_lon_cutoffs, lon)

        if sat_idx == 0:
            sat_idx = 5

        data, lats, lons = all_data[sat_idx - 1]

        mask[i] = get_closest_latlong_sample(data, lats, lons, p)

    return mask

def derive_global_bcm(time, all_data=None, N_lat=1000, N_lon=3000, max_lat=60):
    lat_grid = np.linspace(-max_lat, max_lat, N_lat)
    lon_grid = np.linspace(-180, 180, N_lon)

    lons, lats = np.meshgrid(lon_grid, lat_grid)

    points = np.stack([lats.flatten(), lons.flatten()], axis=1)

    if(all_data is None):
        all_data = [load_bcm(os.path.join(data_dir, f"products/{x}_{time.strftime(dateformat)}.{'grb' if
        ↪  x.startswith('meteosat') else 'nc'}"), lazy_load=True) for x in image_sources.keys()]

    # Sample all points...
    mask = sample_global_bcm(all_data, points).reshape(lats.shape)

    return mask, lats, lons

def load_global_bcm(time, N_lat=1000, N_lon=3000):
    # First check if the derived product exists
    filename = os.path.join(data_dir, f"derived/bcm_{time.strftime(dateformat)}.npz")
    if(os.path.exists(filename)):
        # Load the derived product
        ds = np.load(filename)
        data = ds['BCM']
        lats = ds['Latitude']
        lons = ds['Longitude']

        return data, lats, lons
    else:
        # Load the original products
        data, lats, lons = derive_global_bcm(time, None, N_lat, N_lon)
        # Save the derived product
        np.savez(filename, BCM=data, Latitude=lats, Longitude=lons)
        return data, lats, lons
```

```
# orbits.py
import numpy as np
import datetime
from collections import namedtuple
from typing import Union

from dynamic_tasker.constants import Constants
from dynamic_tasker.rotations import *

def latlong2ecef(latlong):
    lat_deg, lon_deg = latlong
    lat_rad = np.deg2rad(lat_deg)
    lon_rad = np.deg2rad(lon_deg)

```

```
    X = Constants.R_E * np.cos(lat_rad) * np.cos(lon_rad)
    Y = Constants.R_E * np.cos(lat_rad) * np.sin(lon_rad)
    Z = Constants.R_E * np.sin(lat_rad)

    return np.array([X, Y, Z])

def ecef2eci(ecef, time):
    time_difference = time - Constants.J2000
    days_difference = time_difference.total_seconds() / (24 * 60 * 60)
    theta = np.deg2rad((Constants.ERA_J2000 + Constants.gamma * days_difference) % 360)
    rot_mat = rotmat_z(theta)
    return np.dot(rot_mat, ecef)

def eci2ecef(eci, time):
    time_difference = time - Constants.J2000
    days_difference = time_difference.total_seconds() / (24 * 60 * 60)
    theta = np.deg2rad((Constants.ERA_J2000 + Constants.gamma * days_difference) % 360)
    rot_mat = rotmat_z(-theta)
    return np.dot(rot_mat, eci)

def ecef2latlong(ecef):
    X, Y, Z = ecef
    lat = np.arcsin(Z / np.linalg.norm(ecef))
    lon = np.arctan2(Y, X)
    return np.array([np.rad2deg(lat), np.rad2deg(lon)])

def ecef2eci_vec(ecef, time):
    # Ensure ecef is a NumPy array
    ecef = np.asarray(ecef)

    # Check the shape of ecef
    if ecef.ndim != 2 or ecef.shape[1] != 3:
        raise ValueError("ecef must be a 2D array with shape (N, 3)")

    time_difference = time - Constants.J2000
    days_difference = time_difference.total_seconds() / (24 * 60 * 60)

    # Calculate the rotation angle theta in radians
    theta_deg = (Constants.ERA_J2000 + Constants.gamma * days_difference) % 360
    theta = np.deg2rad(theta_deg)

    # Compute the rotation matrix
    rot_mat = rotmat_z(theta)  # Shape: (3, 3)

    # Apply the rotation matrix to all ECEF coordinates
    # Using matrix multiplication: (N, 3) @ (3, 3).T -> (N, 3)
    eci = ecef @ rot_mat.T

    return eci

def latlong2ecef_vec(latlong):
    lat_deg = latlong[:, 0]
    lon_deg = latlong[:, 1]

    # Convert degrees to radians
    lat_rad = np.deg2rad(lat_deg)
    lon_rad = np.deg2rad(lon_deg)

    # Compute ECEF coordinates
    cos_lat = np.cos(lat_rad)
    sin_lat = np.sin(lat_rad)
    cos_lon = np.cos(lon_rad)
    sin_lon = np.sin(lon_rad)

    X = Constants.R_E * cos_lat * cos_lon
    Y = Constants.R_E * cos_lat * sin_lon
    Z = Constants.R_E * sin_lat

```

```
    # Stack into a (N, 3) array
    ecef = np.column_stack((X, Y, Z))

    return ecef

def latlong2eci(lat, long, time):
    # Convert latlong to ECEF coordinates
    ecef = latlong2ecef((lat, long))

    # Convert ECEF to ECI coordinates
    eci = ecef2eci(ecef, time)

    return eci

# Calculate the vector pointing to the sun in the ECI frame
def sunvec_eci(time):
    time_difference = time - Constants.J2000
    d = time_difference.total_seconds() / (24 * 60 * 60)
    L = 280.4606184 + ((36000.77005361 / 36525) * d) # mean longitude
    g = 357.5277233 + ((35999.05034 / 36525) * d) # mean anomaly
    p = L + (1.914666471 * np.sin(g * np.pi / 180)) + (0.918994643 * np.sin(2*g * np.pi / 180)) # ecliptic
    ↪  longitude lambda
    q = 23.43929 - ((46.8093/3600) * (d / 36525)) # obliquity of the ecliptic plane epsilon
    return np.array((np.cos(p * np.pi / 180), np.cos(q * np.pi / 180) * np.sin(p * np.pi / 180), np.sin(q * np.pi
    ↪  / 180) * np.sin(p * np.pi / 180)))


# Define a namedtuple for Keplerian elements
Keplerian = namedtuple('Keplerian', ['a', 'e', 'i', 'omega', 'Omega', 'M', 't'])

# Six keplerian elements: a, e, i, omega, Omega, M
def kepler2eci(elements: Keplerian) -> tuple:
    a, e, i, omega, Omega, M, t = elements
    mu = Constants.mu
    nu = None
    r = None
    if(e != 0):
        # Solve Kepler's Equation for E (Eccentric Anomaly)
        E = M
        error = 1
        while error > 1e-6:
            E_new = M + e * np.sin(E)
            error = np.abs(E_new - E)
            E = E_new

        # True anomaly
        nu = 2 * np.arctan(np.sqrt((1 + e) / (1 - e)) * np.tan(E / 2))
        # Distance (radius) from the central body
        r = a * (1 - e * np.cos(E))
    else:
        nu = M
        r = a

    # Position in the perifocal coordinate system
    r_perifocal = np.array([r * np.cos(nu), r * np.sin(nu), 0])

    # Velocity in the perifocal coordinate system
    h = np.sqrt(mu * a * (1 - e**2))
    v_perifocal = np.array([-mu / h * np.sin(nu), mu / h * (e + np.cos(nu)), 0])

    # Rotation matrices
    R_Omega = np.array([
        [np.cos(Omega), -np.sin(Omega), 0],
        [np.sin(Omega), np.cos(Omega), 0],
        [0, 0, 1]
    ])
    R_i = np.array([
        [1, 0, 0],
```

```
        [0, np.cos(i), -np.sin(i)],
        [0, np.sin(i), np.cos(i)]
    ])
    R_omega = np.array([
        [np.cos(omega), -np.sin(omega), 0],
        [np.sin(omega), np.cos(omega), 0],
        [0, 0, 1]
    ])

    # Complete rotation matrix from perifocal to ECI
    R = R_Omega @ R_i @ R_omega

    # Position and velocity in the ECI frame
    r_eci = R @ r_perifocal
    v_eci = R @ v_perifocal

    return r_eci, v_eci

# Generate a circular orbit
def circular_orbit(a: float, i: float, Omega: float, M: float, t: datetime) -> Keplerian:
    return Keplerian(a, 0, i, 0, Omega, M, t)

def propagate_orbit(orbit: Keplerian, time: Union[float, datetime.datetime, datetime.timedelta]) -> Keplerian:
    mu = Constants.mu
    a, e, i, omega, Omega, M, t = orbit
    n = np.sqrt(mu / a**3)
    if(isinstance(time, datetime.timedelta)):
        time = time.total_seconds()
    elif(isinstance(time, datetime.datetime)):
        time = (time - t).total_seconds()

    M_new = M + n * time
    return Keplerian(a, e, i, omega, Omega, M_new, t + datetime.timedelta(seconds=time))

def v_orb(h):
    return np.sqrt(Constants.mu / (h + Constants.R_E))

def t_orb(elements: Keplerian):
    return 2 * np.pi * np.sqrt((elements.a**3) / Constants.mu)

def horizon_distance(elements):
    return np.sqrt((elements.a)**2 - Constants.R_E**2)

def horizon_angle(elements):
    return np.arcsin(Constants.R_E / elements.a)

def horizon_time(elements):
    return horizon_spherical_angle(elements) * elements.a / (v_orb(elements.a - Constants.R_E))

def horizon_spherical_angle(elements):
    return np.arccos(Constants.R_E / elements.a)

def intersect_ray_sphere(P, u, x0, r, horizon_snap=False):
    """
    Intersect (or tangent-snap) a ray P + t u with a sphere of centre x0, radius r.

    Returns
    -------
    (pt1, pt2, t1, t2)
        • Two 3-D points (may be the same) and their parameter values along the ray.
        • If the ray does not meet the sphere and horizon_snap is False → None.
        • If horizon_snap is True and there is no intersection → the tangent
          point is returned twice, with the appropriate single parameter t.
    """
    P   = np.asarray(P, dtype=float)
    u   = np.asarray(u, dtype=float)
    x0  = np.asarray(x0, dtype=float)
    if np.allclose(u, 0):
```

```
        raise ValueError("Direction vector u must be non-zero")

    # Normalise the direction so that 't' is in metres (or whatever units you use)
    u   = u / np.linalg.norm(u)

    # Quadratic coefficients for |P + t u − x0|² = r²
    d   = P - x0
    A   = 1.0                           # because u is unit
    B   = 2.0 * np.dot(u, d)
    C   = np.dot(d, d) - r*r
    disc = B*B - 4*A*C

    # ---------------- regular intersection cases ----------------
    if disc > 0:                        # two points
        sqrt_disc = np.sqrt(disc)
        t1 = (-B + sqrt_disc) / (2*A)
        t2 = (-B - sqrt_disc) / (2*A)
        return (P + t1*u, P + t2*u, t1, t2)

    if np.isclose(disc, 0):             # exactly tangent already
        t = -B / (2*A)
        pt = P + t*u
        return (pt, pt, t, t)

    # ---------------- no hit, maybe horizon-snap? ----------------
    if not horizon_snap:
        return None

    # ---- compute tangent point in the plane spanned by d and u ----
    L2 = np.dot(d, d)
    if L2 <= r*r:
        # P happens to lie on or inside the sphere − snapping undefined
        return None

    # Plane normal and an in-plane unit vector perpendicular to d
    n = np.cross(u, d)
    if np.linalg.norm(n) < 1e-12:       # u ∥ d    choose any perpendicular axis
        n = np.cross(d, np.array([1.0, 0.0, 0.0]))
        if np.linalg.norm(n) < 1e-12:   # unlucky − d happens to be x-axis
            n = np.cross(d, np.array([0.0, 1.0, 0.0]))
    k = np.cross(n, d)
    k_hat = k / np.linalg.norm(k)

    L    = np.sqrt(L2)
    delta = np.sqrt(L2 - r*r)

    # Two candidate tangent points
    T1 = x0 + (r*r / L2) * d + (r * delta / L) * k_hat
    T2 = x0 + (r*r / L2) * d - (r * delta / L) * k_hat

    # Choose the one that lies "forward" along u (positive dot)
    dir1 = (T1 - P)
    dir2 = (T2 - P)
    dot1 = -np.dot(dir1, u)
    dot2 = -np.dot(dir2, u)

    if dot1 >= 0 and dot2 < 0:
        T = T1; t = -dot1                # u is unit, so t = proj length
    elif dot2 >= 0 and dot1 < 0:
        T = T2; t = -dot2
    else:
        # Both are forward (rare) − pick the smaller angular offset
        ang1 = np.arccos(dot1 / np.linalg.norm(dir1))
        ang2 = np.arccos(dot2 / np.linalg.norm(dir2))
        T, t = (T1, dot1) if ang1 < ang2 else (T2, dot2)

    return (T, T, t, t)

```

```python
def earth_line_intersection(P, u, horizon_snap=False):
    p = intersect_ray_sphere(P, u, np.array([0, 0, 0]), Constants.R_E, horizon_snap)
    if(p is None):
        return p
    else:
        p1, p2, t1, t2 = p
        if(t1 < 0 and t2 < 0):
            return (p1, p2)
        else:
            return None

def split_orbit_track(latlongs, threshold=180):
    lat, long = np.array(latlongs)[:, 0], np.array(latlongs)[:, 1]
    # Calculate the difference between consecutive longitudes
    delta_long = np.abs(np.diff(long))

    # Identify where the jump exceeds the threshold
    jump_indices = np.where(delta_long > threshold)[0] + 1

    # Split the data at the jump indices
    segments = np.split(latlongs, jump_indices)
    return segments

def kepler2latlong(orbit: Keplerian, time):
    return ecef2latlong(eci2ecef(kepler2eci(propagate_orbit(orbit, (time - orbit.t).total_seconds()))[0], time))
```

```python
# rotations.py
import numpy as np

def qconj(q):
    p = q.copy()
    p[1:] *= -1
    return p

def qmult(q1, q2):
    w1, v1 = q1[0], q1[1:]
    w2, v2 = q2[0], q2[1:]
    # Use geometric product to multiply quaternions
    w = w1*w2 - np.dot(v1, v2)
    v = w1*v2 + w2*v1 + np.cross(v1, v2)
    return np.hstack([w, v])

def qsandwich(q, v):
    p = np.hstack([0, v])
    return qmult(q, qmult(p, qconj(q)))[1:]

def qarray(q, func, *args):
    return np.array([func(qi, *args) for qi in q])

def q2mat(q, homogenous=False):
    w, x, y, z = q
    if(not homogenous):
        return np.array([
            [1 - 2*y**2 - 2*z**2, 2*x*y - 2*z*w, 2*x*z + 2*y*w],
            [2*x*y + 2*z*w, 1 - 2*x**2 - 2*z**2, 2*y*z - 2*x*w],
            [2*x*z - 2*y*w, 2*y*z + 2*x*w, 1 - 2*x**2 - 2*y**2]
        ])

    return np.array([
        [1 - 2*y**2 - 2*z**2, 2*x*y - 2*z*w, 2*x*z + 2*y*w, 0],
        [2*x*y + 2*z*w, 1 - 2*x**2 - 2*z**2, 2*y*z - 2*x*w, 0],
        [2*x*z - 2*y*w, 2*y*z + 2*x*w, 1 - 2*x**2 - 2*y**2, 0],
        [0, 0, 0, 1]
    ])

def eul2q(yaw, pitch, roll):
    # Calculate the half angles
    cy = np.cos(yaw * 0.5)
```

```
    sy = np.sin(yaw * 0.5)
    cp = np.cos(pitch * 0.5)
    sp = np.sin(pitch * 0.5)
    cr = np.cos(roll * 0.5)
    sr = np.sin(roll * 0.5)

    # Calculate the quaternion
    w = cr * cp * cy + sr * sp * sy
    x = sr * cp * cy - cr * sp * sy
    y = cr * sp * cy + sr * cp * sy
    z = cr * cp * sy - sr * sp * cy

    return (w, x, y, z)

def rotmat_x(theta):
    return np.array([[np.cos(theta), -np.sin(theta), 0],
                     [np.sin(theta), np.cos(theta), 0],
                     [0, 0, 1]])

def rotmat_y(theta):
    return np.array([[np.cos(theta), -np.sin(theta), 0],
                     [np.sin(theta), np.cos(theta), 0],
                     [0, 0, 1]])

def rotmat_z(theta):
    return np.array([[np.cos(theta), -np.sin(theta), 0],
                     [np.sin(theta), np.cos(theta), 0],
                     [0, 0, 1]])

def eul2R(roll, pitch, yaw):

    # Calculate rotation matrix for roll (around x-axis)
    R_x = np.array([[1, 0, 0],
                    [0, np.cos(roll), -np.sin(roll)],
                    [0, np.sin(roll), np.cos(roll)]])

    # Calculate rotation matrix for pitch (around y-axis)
    R_y = np.array([[np.cos(pitch), 0, np.sin(pitch)],
                    [0, 1, 0],
                    [-np.sin(pitch), 0, np.cos(pitch)]])

    # Calculate rotation matrix for yaw (around z-axis)
    R_z = np.array([[np.cos(yaw), -np.sin(yaw), 0],
                    [np.sin(yaw), np.cos(yaw), 0],
                    [0, 0, 1]])

    # Combine the rotations
    R = R_z @ (R_y @ R_x)

    return R

def rotmat_from_vec(a: np.ndarray, b: np.ndarray) -> np.ndarray:
    """
    Computes the rotation matrix that rotates the standard basis vectors
    x = [1, 0, 0] and y = [0, 1, 0] to the provided orthogonal vectors a and b.

    Parameters:
    a (np.ndarray): The target vector for the x-axis (should be a 3-element array).
    b (np.ndarray): The target vector for the y-axis (should be a 3-element array).

    Returns:
    np.ndarray: A 3x3 rotation matrix.

    Raises:
    ValueError: If the input vectors are not 3-dimensional or not orthogonal.
    """
    # Ensure input vectors are numpy arrays
    a = np.asarray(a, dtype=float)
```

```
    b = np.asarray(b, dtype=float)

    # Check if vectors are 3-dimensional
    if a.shape != (3,) or b.shape != (3,):
        raise ValueError("Input vectors must be three-dimensional.")

    # Normalize the vectors
    a_norm = np.linalg.norm(a)
    b_norm = np.linalg.norm(b)

    if a_norm == 0 or b_norm == 0:
        raise ValueError("Input vectors must be non-zero.")

    a_unit = a / a_norm
    b_unit = b / b_norm

    # Check orthogonality
    dot_product = np.dot(a_unit, b_unit)
    if not np.isclose(dot_product, 0.0, atol=1e-8):
        raise ValueError("Input vectors must be orthogonal.")

    # Compute the third orthogonal vector using cross product
    c_unit = np.cross(a_unit, b_unit)

    # Form the rotation matrix with a_unit, b_unit, c_unit as columns
    R = np.column_stack((a_unit, b_unit, c_unit))

    # Verify that R is a valid rotation matrix
    if not np.allclose(np.dot(R, R.T), np.identity(3), atol=1e-8):
        raise ValueError("Resulting matrix is not orthogonal.")
    if not np.isclose(np.linalg.det(R), 1.0, atol=1e-8):
        raise ValueError("Resulting matrix does not have a determinant of +1.")

    return R
```

```
# schedulers.py
import datetime
import re
import os
import numpy as np
from pyscipopt import Model, quicksum
from dataclasses import dataclass
from typing import List, Callable
from pathlib import Path
from dynamic_tasker.access import Request
from dynamic_tasker.orbits import Keplerian

data_dir = Path(__file__).resolve().parent.parent.parent / "data"

@dataclass
class Spacecraft:
    orbit: Keplerian
    agility: Callable
    # TODO: Maybe add instrument types?

@dataclass
class Scenario:
    requests: List[Request]
    satellites: List[Spacecraft]

def load_worldcities(n=1000, random_subsample=False):
    accesses = []
    # Open the CSV and process the lines
    with open(os.path.join(data_dir, 'worldcities/worldcities.csv'), 'r', encoding='utf8') as f:
        f.readline()  # Skip the header line
        i = 0
        for line in f:
            parts = re.split(r'(?:",")|"', line)
```

```
            # Extract the latitude, longitude, and city name
            lat = float(parts[3])
            lon = float(parts[4])
            city = parts[2]
            # Create a new Request object and append it to the list
            accesses.append(Request(len(accesses), lat, lon, city))
            i += 1
            if i >= n:
                break

    return accesses

# def save_scenario(scenario, filename):
    # Use pickle


def greedy_schedule(accesses, requests, agility):
    schedule = []
    request_mask = np.zeros(len(requests))
    for i in accesses:
        if(len(schedule) == 0):
            schedule.append(i)
        else:
            if(request_mask[i.requestid] == 0 and i.time > datetime.timedelta(seconds=agility(i.angle -
            ↪  schedule[-1].angle)) + schedule[-1].time):
                schedule.append(i)
                request_mask[i.requestid] = 1

    return schedule

def milp_schedule(accesses, requests, agility, force_in_schedule=None):
    model = Model("Scheduler")
    model.hideOutput()
    x = {}
    # Map of index to requestid
    idx_map = []

    for i, a in enumerate(accesses):
        x[i] = model.addVar(vtype="B", name=f"x_{a.requestid}_{a.time}")
        if(force_in_schedule is not None and a in force_in_schedule):
            model.addCons(x[i] == 1)

        idx_map.append(a.requestid)

    # Add constraints based on agility and repetition
    for i in range(len(accesses) - 1):
        for j in range(i + 1, len(accesses)):
            if(idx_map[i] == idx_map[j] or accesses[j].time <
            ↪  datetime.timedelta(seconds=agility(accesses[j].angle - accesses[i].angle)) + accesses[i].time):
                model.addCons(x[i] + x[j] <= 1)

    # Add objective
    model.setObjective(quicksum(x[i] * a.utility for i, a in enumerate(accesses)), "maximize")
    res = model.optimize()
    sol = model.getBestSol()
    schedule = []
    for i, a in enumerate(accesses):
        if(sol[x[i]] == 1):
            schedule.append(a)

    return schedule

# Constellation scheduler
# No forces in for now
def milp_schedule_constellation(accesses_all, requests, agility):
    # Get the number of satellites
    n_satellites = len(accesses_all)
    model = Model("Scheduler_Constellation")
```

```
    idx_map = []

    x = {}
    current_idx = 0
    for j, accesses in enumerate(accesses_all):
        for a in accesses:
            x[current_idx] = model.addVar(vtype="B", name=f"x_sat{j}_{a.requestid}_{a.time}")
            current_idx += 1

            idx_map.append(a.requestid)

    # Add constraints based on agility
    for k in range(len(accesses_all)):
        start_idx = np.sum([len(accesses_all[i]) for i in range(k)])
        for i in range(len(accesses_all[k]) - 1):
            for j in range(i + 1, len(accesses_all[k])):
                if(accesses_all[k][j].time < datetime.timedelta(seconds=agility(accesses_all[k][j].angle -
                ↪  accesses_all[k][i].angle)) + accesses_all[k][i].time):
                    model.addCons(x[start_idx + i] + x[start_idx + j] <= 1)

    # Add constraints based on repetition
    for i in range(len(idx_map)):
        for j in range(i + 1, len(idx_map)):
            if(idx_map[i] == idx_map[j]):
                model.addCons(x[i] + x[j] <= 1)

    # Add objective
    model.setObjective(quicksum(x[i] * a.utility for i in range(len(idx_map))), "maximize")
    res = model.optimize()
    sol = model.getBestSol()
    schedule = [[] for _ in range(len(accesses_all))]
    for k in range(len(accesses_all)):
        start_idx = int(np.sum([len(accesses_all[i]) for i in range(k)]))
        for i in range(len(accesses_all[k])):
            if(sol[x[start_idx + i]] == 1):
                schedule[k].append(accesses_all[k][i])

    return schedule

def no_repair(tasks, requests, agility):
    return tasks

# TODO: get rid of the occluded mask
def greedy_scheduleрepair(schedule, total_tasks, requests, occluded_mask, agility, allow_duplicates=False):
    schedule_copy = schedule.copy()
    request_mask = np.zeros(len(requests))
    for i in schedule:
        request_mask[i.requestid] = 1

    stop_iterating = False
    i = 0
    while(not stop_iterating):
        if(i == len(schedule_copy) - 1):
            break

        # Current node, looking ahead
        current = schedule_copy[i]
        next = schedule_copy[i + 1]
        next_next = schedule_copy[i + 2] if i + 2 < len(schedule_copy) else None

        total_index = total_tasks.index(next)
        if(occluded_mask[total_index] == 1):
            # Find the next non-occluded task
            next_nonoccluded = [x for x in total_tasks[i:] if occluded_mask[total_tasks.index(x)] == 0 and x.time
            ↪  > current.time]

            if(next_next):
                next_nonoccluded = [x for x in next_nonoccluded if x.time < next_next.time]
```

```
166
167             if(len(next_nonoccluded) == 0):
168                 schedule_copy.pop(i)
169                 break
170
171             for j in next_nonoccluded:
172                 if(j.time > datetime.timedelta(seconds=agility(j.angle - current.angle)) + current.time and
                    ↪  (next_next) and
173                     next_next.time > datetime.timedelta(seconds=agility(next_next.angle - j.angle)) + j.time):
174                     if(not allow_duplicates):
175                         if(request_mask[j.requestid] == 1):
176                             continue
177
178                     schedule_copy.pop(i + 1)
179                     schedule_copy.insert(i + 1, j)
180                     request_mask[j.requestid] = 1
181
182         i += 1
183
184     return schedule_copy
185
186 def slew_angle(t, t_next, angle_origin, angle_dest, agility):
187     t_total = agility(angle_dest - angle_origin)
188     t_s = agility(0)
189     t_norm = (t_total - t_next + t)/(t_total - t_s)
190     theta = angle_dest - angle_origin
191     t_untilnext = t_next - t
192     if (t_untilnext <= t_s):
193         return angle_dest
194     elif (t_untilnext >= t_total):
195         return angle_origin
196     elif (t_norm <= 0.5):
197         return 2 * theta * t_norm**2 + angle_origin
198     else:
199         return theta * (1 - 2 * (1 - t_norm)**2) + angle_origin
200
201 def temporal_slew_angle(t_now, prev, next, agility):
202     return slew_angle(t_now, next.time, prev.angle, next.angle, agility)
203
204 def eval_scenario(scenario, initial_scheduler, repair_scheduler):
205     pass
```

Appendix B

Revision History

Table B.1: Revision history of this document.

Date	Description	Sections Affected
2025-08-04	Initial draft submission	All
2025-08-29	Revisions submitted	All

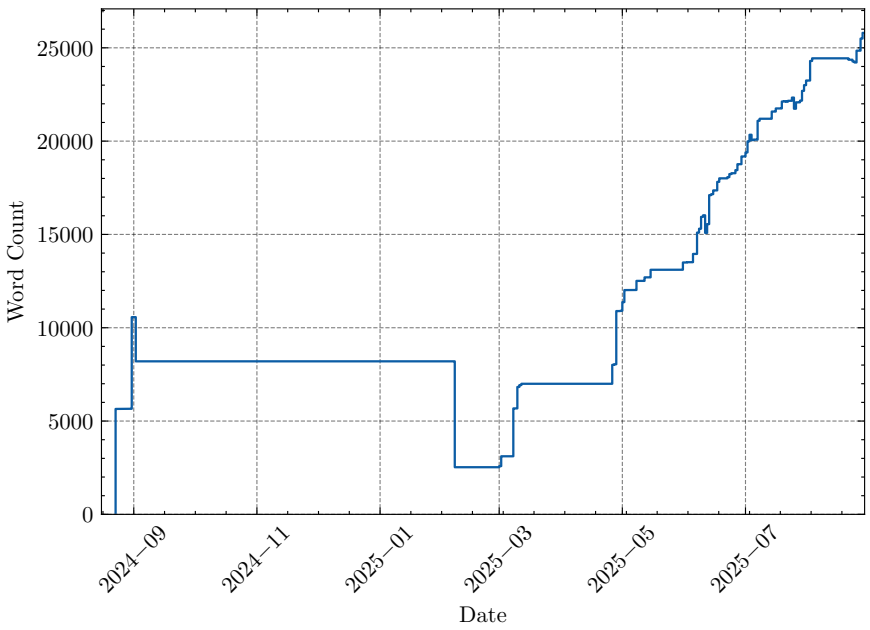


Figure B.1: Word count over time while writing this document.

References

[1] Planet | Insights - Our Constellations, April 2021. URL https://planet.com/insights/.

[2] BlackSky, . URL https://www.blacksky.com/.

[3] Satellogic, . URL https://satellogic.com/.

[4] Michael D. King, Steven Platnick, W. Paul Menzel, Steven A. Ackerman, and Paul A. Hubanks. Spatial and Temporal Distribution of Clouds Observed by MODIS Onboard the Terra and Aqua Satellites. *IEEE Transactions on Geoscience and Remote Sensing*, 51 (7):3826–3852, July 2013. ISSN 1558-0644. doi:10.1109/TGRS.2012.2227333. URL https://ieeexplore.ieee.org/document/6422379. Conference Name: IEEE Transactions on Geoscience and Remote Sensing.

[5] NASA - NSSDCA - Spacecraft - Telemetry Details, . URL https://nssdc.gsfc.nasa.gov/nmc/spacecraft/displayTrajectory.action?id=1959-004A.

[6] NASA - NSSDCA - Spacecraft - Details, . URL https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1959-001A.

[7] NOAA Digital Collections | National Oceanic and Atmospheric Administration, . URL https://www.noaa.gov/digital-collections.

[8] Theresa B. Tabak. *Eyes in the Sky: Eisenhower, the CIA and Cold War Aerial Espionage*. Naval Institute Press, Annapolis, Md, 2010. ISBN 978-1-59114-082-5.

[9] Kevin Conley Ruffner. *Corona: America's first satellite program*. Morgan James, New York, 2011. ISBN 978-0-9758570-4-5. OCLC: 772235331.

[10] NRO – Corona History, . URL https://www.nro.gov/About-NRO/history/history-corona/.

[11] Landsat 1 | Landsat Science, November 2021. URL https://landsat.gsfc.nasa.gov/satellites/landsat-1/.

[12] Landsat, What's Next?, . URL https://www.congress.gov/crs-product/IN12281.

[13] Dennis Flood. NASA Photovoltaic Research and Technology.

[14] Duncan Eddy. *Task Planning for Earth Observing Satellite Systems*. PhD thesis, Stanford University, 2021. URL https://stacks.stanford.edu/file/druid:fp397ds6833/eddy_thesis_final-augmented.pdf.

[15] National Space Policy Directive 42, 1982. URL https://www.reaganlibrary.gov/public/archives/reference/scanned-nsdds/nsdd42.pdf.

[16] SPOT 1 Objectives - Earth Online, . URL https://earth.esa.int/eogateway/missions/spot-1/objectives.

[17] George E. Rep. Brown. H.R.6133 - 102nd Congress (1991-1992): Land Remote Sensing Policy Act of 1992, October 1992. URL https://www.congress.gov/bill/102nd-congress/house-bill/6133. Archive Location: 1992-10-05.

[18] WorldView Legion | Maxar, . URL https://www.maxar.com/worldview-legion.

[19] WorldView-1 | Specifications Of WorldView-1 Sensor, . URL https://satpalda.co/worldview-1/.

[20] WorldView-3 Satellite Sensor | Satellite Imaging Corp, . URL https://www.satimagingcorp.com/satellite-sensors/worldview-3/.

[21] Sylvain Damiani, Gérard Verfaillie, and Marie-Claire Charmeau. A Continuous Anytime Planning Module for an Autonomous Earth Watching Satellite. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling*, 2005. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4b3b5f8df89e8748c6f69aa2b14cf70d3e5c4db4#page=23.

[22] Michel Lemaıtre, Gérard 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. ISSN 12709638. doi:10.1016/S1270-9638(02)01173-2. URL https://linkinghub.elsevier.com/retrieve/pii/S1270963802011732.

[23] Landsat Data Access | U.S. Geological Survey, January 2018. URL https://www.usgs.gov/landsat-missions/landsat-data-access.

[24] Origins of Google Earth : History of Information, . URL https://www.historyofinformation.com/detail.php?id=2733.

[25] Yuhan Wang, Qian Zhang, Feng Yu, Na Zhang, Xining Zhang, Yuchen Li, Ming Wang, and Jinmeng Zhang. Progress in Research on Deep Learning-Based Crop Yield Prediction. *Agronomy*, 14(10):2264, October 2024. ISSN 2073-4395. doi:10.3390/agronomy14102264. URL https://www.mdpi.com/2073-4395/14/10/2264. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute.

[26] Szymon Glinka. Sat4BIM4D — the concept of using satellite remote sensing to monitor construction progress in conjunction with BIM. *Reports on Geodesy and Geoinformatics*, 118(1), December 2024. doi:10.2478/rgg-2024-0023. URL https://sciendo.com/article/10.2478/rgg-2024-0023.

[27] Global Fishing Watch. New research harnesses AI and satellite imagery to reveal the expanding footprint of human activity at sea, January 2024. URL https://globalfishingwatch.org/press-release/new-research-harnesses-ai-and-satellite-imagery-to-reveal-the-expanding-footprint-of-human-activity

[28] Kodai Yasuda, Ryuichi Shibasaki, Riku Yasuda, and Hiroki Murata. Terminal Congestion Analysis of Container Ports Using Satellite Images and AIS. *Remote Sensing*, 16 (6):1082, January 2024. ISSN 2072-4292. doi:10.3390/rs16061082. URL https://www.mdpi.com/2072-4292/16/6/1082. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[29] Fatih Fehmi Şimşek. Determination of land use and land cover change using multi-temporal PlanetScope images and deep learning CNN model. *Paddy and Water Environment*, May 2025. ISSN 1611-2504. doi:10.1007/s10333-025-01024-9. URL https://doi.org/10.1007/s10333-025-01024-9.

[30] Michele Finizio, Federica Pontieri, Chiara Bottaro, Mirko Di Febbraro, Michele Innangi, Giovanna Sona, and Maria Laura Carranza. Remote Sensing for Urban Biodiversity: A Review and Meta-Analysis. *Remote Sensing*, 16(23):4483, January 2024. ISSN 2072-4292. doi:10.3390/rs16234483. URL https://www.mdpi.com/2072-4292/16/23/4483. Number: 23 Publisher: Multidisciplinary Digital Publishing Institute.

[31] Afonso Henrique Moraes Oliveira, José Humberto Chaves, Eraldo Aparecido T. Matricardi, Iara Musse Felix, Mauro Mendonça Magliano, and Lucietta Guerreiro Martorano. Monitoring sustainable forest management plans in the Amazon: Integrating LiDAR data and PlanetScope imagery. *Remote Sensing Applications: Society and Environment*, 38:101535, April 2025. ISSN 2352-9385. doi:10.1016/j.rsase.2025.101535. URL https://www.sciencedirect.com/science/article/pii/S2352938525000886.

[32] Jörg Brauchle, Matthias Geßner, Thomas Kraft, Daniel Hein, Michael Lesmeister, Julia Gonschorek, Marius Bock, and Ralf Berger. Regional Rapid Mapping for First Responders - Turkey 2023 Earthquake. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVIII-3/W3-2024:15–19, November 2024. ISSN 2194-9034. doi:10.5194/isprs-archives-XLVIII-3-W3-2024-15-2024. URL https://isprs-archives.copernicus.org/articles/XLVIII-3-W3-2024/15/2024/.

[33] Bertrand Rouet-Leduc and Claudia Hulbert. Automatic detection of methane emissions in multispectral satellite imagery using a vision transformer. *Nature Communications*, 15(1):3801, May 2024. ISSN 2041-1723. doi:10.1038/s41467-024-47754-y. URL https://www.nature.com/articles/s41467-024-47754-y. Publisher: Nature Publishing Group.

[34] First Greenhouse Gas Plumes Detected With NASA-Designed Instrument - NASA, October 2024. URL https://www.nasa.gov/earth/first-greenhouse-gas-plumes-detected-with-nasa-designed-instrument/. Section: Earth.

[35] Theresa Thompson Chaudhry, Mahnoor Asif, and Mirza Manuchehar Hussain. Using Nighttime Light Data to Estimate Sub-national GDP and Growth in Pakistan, May 2024. URL https://papers.ssrn.com/abstract=4814807.

[36] Jeff Higgins, Usman Adamu, Kehinde Adewara, Adeshina Aladeshawe, Aron Aregay, Inuwa Barau, Andrew Berens, Omotayo Bolu, Nina Dutton, Nnaemeka Iduma, Bryant Jones, Brian Kaplan, Sule Meleh, Melton Musa, Gatei wa Nganda, Vincent Seaman, Anupma Sud, Stephane Vouillamoz, and Eric Wiesen. Finding inhabited settlements and tracking vaccination progress: the application of satellite imagery analysis to guide the immunization response to confirmation of previously-undetected, ongoing endemic wild poliovirus transmission in Borno State, Nigeria. *International Journal of Health Geographics*, 18(1):11, May 2019. ISSN 1476-072X. doi:10.1186/s12942-019-0175-y. URL https://doi.org/10.1186/s12942-019-0175-y.

[37] Zsolt Katona, Marcus Painter, Panos N. Patatoukas, and Jean (Jieyin) Zeng. On the Capital Market Consequences of Big Data: Evidence from Outer Space, July 2018. URL https://papers.ssrn.com/abstract=3222741.

[38] Kieran P. Fitzmaurice, Helena M. Garcia, Antonia Sebastian, Hope Thomson, Harrison B. Zeff, and Gregory W. Characklis. Flood risks to the financial stability of residential mortgage borrowers: An integrated modeling approach. *EGUsphere*, pages 1–40, May 2025. doi:10.5194/egusphere-2025-2049. URL https://egusphere.copernicus.org/preprints/2025/egusphere-2025-2049/. Publisher: Copernicus GmbH.

[39] Kiri L. Wagstaff, Terran Lane, and Alex Roper. Multiple-Instance Regression with Structured Data. In *2008 IEEE International Conference on Data Mining Workshops*, pages 291–300, December 2008. doi:10.1109/ICDMW.2008.31. URL https://ieeexplore.ieee.org/document/4733948/. ISSN: 2375-9259.

[40] Y. Sun, C. Frankenberg, J. D. Wood, D. S. Schimel, M. Jung, L. Guanter, D. T. Drewry, M. Verma, A. Porcar-Castell, T. J. Griffis, L. Gu, T. S. Magney, P. Köhler, B. Evans, and K. Yuen. OCO-2 advances photosynthesis observation from space via solar-induced chlorophyll fluorescence. *Science*, 358(6360):eaam5747, October 2017. ISSN 0036-8075, 1095-9203. doi:10.1126/science.aam5747. URL https://www.science.org/doi/10.1126/science.aam5747.

[41] Landsat Missions | U.S. Geological Survey, . URL https://www.usgs.gov/landsat-missions.

[42] M. Drusch, U. Del Bello, S. Carlier, O. Colin, V. Fernandez, F. Gascon, B. Hoersch, C. Isola, P. Laberinti, P. Martimort, A. Meygret, F. Spoto, O. Sy, F. Marchese, and P. Bargellini. Sentinel-2: ESA's Optical High-Resolution Mission for GMES Operational Services. *Remote Sensing of Environment*, 120:25–36, May 2012. ISSN 0034-4257. doi:10.1016/j.rse.2011.11.026. URL https://www.sciencedirect.com/science/article/pii/S0034425712000636.

[43] C.L. Parkinson. Aqua: an Earth-Observing Satellite mission to examine water and other climate variables. *IEEE Transactions on Geoscience and Remote Sensing*, 41(2):

173–183, February 2003. ISSN 1558-0644. doi:10.1109/TGRS.2002.808319. URL https://ieeexplore.ieee.org/document/1196036. Conference Name: IEEE Transactions on Geoscience and Remote Sensing.

[44] Angelita Kelly, Eric Moyer, Dimitrios Mantziaras, and Warren Case. Terra mission operations: Launch to the present (and beyond). page 92180M, San Diego, California, United States, September 2014. doi:10.1117/12.2061253. URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2061253.

[45] S.A. McDermott, A. Jacobovits, and H. Yashiro. Automotive electronics in space: combining the advantages of high reliability components with high production volume. In *Proceedings, IEEE Aerospace Conference*, volume 4, pages 4–4, March 2002. doi:10.1109/AERO.2002.1036898. URL https://ieeexplore.ieee.org/document/1036898.

[46] Harry W. Jones. The Recent Large Reduction in Space Launch Cost. Stafford Springs, CT, United States, July 2018. URL https://ntrs.nasa.gov/citations/20200001093. NTRS Author Affiliations: NASA Ames Research Center NTRS Report/Patent Number: ARC-E-DAA-TN56851 NTRS Document ID: 20200001093 NTRS Research Center: Ames Research Center (ARC).

[47] Capella Space, . URL https://www.capellaspace.com.

[48] Michael Nolde, Simon Plank, Rudolf Richter, Doris Klein, and Torsten Riedlinger. The DLR FireBIRD Small Satellite Mission: Evaluation of Infrared Data for Wildfire Assessment. *Remote Sensing*, 13(8):1459, January 2021. ISSN 2072-4292. doi:10.3390/rs13081459. URL https://www.mdpi.com/2072-4292/13/8/1459. Number: 8 Publisher: Multidisciplinary Digital Publishing Institute.

[49] Christoph Lenzen, Maria T. Woerle, Tobias Göttfert, Falk Mrowka, and Martin Wickler. Onboard Planning and Scheduling Autonomy within the Scope of the FireBird Mission. In *SpaceOps 2014 Conference*, Pasadena, CA, May 2014. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-221-9. doi:10.2514/6.2014-1759. URL https://arc.aiaa.org/doi/10.2514/6.2014-1759.

[50] Jason Swope, Steve Chien, Xavier Bosch-Lluis, Emily Dunkel, Qing Yue, Mehmet Ogut, Isaac Ramos, Pekka Kangaslahti, William Deal, and Caitlyn Cooke. Storm Classification and Dynamic Targeting for a Smart Ice Cloud Sensing Satellite. *Journal of Aerospace Information Systems*, 21(12):958–971, December 2024. ISSN 1940-3151, 2327-3097. doi:10.2514/1.I011318. URL https://arc.aiaa.org/doi/10.2514/1.I011318.

[51] Alberto Candela, Juan Delfa Victoria, Itai Zilberstein, Marcin Kurowski, Qing Yue, and Steve Chien. Dynamic Targeting Scenario to Study the Planetary Boundary Layer. *International Geoscience and Remote Sensing Symposium*, 2024.

[52] Planetary Variables: Quantifying a Changing World | Planet, . URL https://www.planet.com/products/planetary-variables/.

[53] Planet Labs PBC Announces Real-Time Insights Technology Using NVIDIA Jetson Platform, 2024. URL https://investors.planet.com/news/news-details/2024/Planet-Labs-PBC-Announces-Real-Time-Insights-Technology-Using-NVIDIA-Jetson-Platform/default.aspx.

[54] Shreeyam Kacker, Alex Meredith, Kerri Cahoy, and Georges Labrèche. Machine Learning Image Processing Algorithms Onboard OPS-SAT. *Small Satellite Conference*, August 2022. URL https://digitalcommons.usu.edu/smallsat/2022/all2022/65.

[55] Sorour Mohajerani and Parvaneh Saeedi. *Cloud-Net+: A Cloud Segmentation CNN for Landsat 8 Remote Sensing Imagery Optimized with Filtered Jaccard Loss Function*. January 2020.

[56] Sorour Mohajerani and Parvaneh Saeedi. Cloud-Net: An End-To-End Cloud Detection Algorithm for Landsat 8 Imagery. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 1029–1032, July 2019. doi:10.1109/IGARSS.2019.8898776. URL https://ieeexplore.ieee.org/document/8898776/. ISSN: 2153-7003.

[57] M. Schmitt and Y.-L. Wu. Remote Sensing Image Classification with the SEN12MS Dataset. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-2-2021:101–106, June 2021. ISSN 2194-9042. doi:10.5194/isprs-annals-V-2-2021-101-2021. URL https://isprs-annals.copernicus.org/articles/V-2-2021/101/2021/. Conference Name: XXIV ISPRS Congress <q>Imaging today, foreseeing tomorrow</q>, Commission II - 2021 edition, 5–9 July 2021 Publisher: Copernicus GmbH.

[58] Aman Agarwal, James Gearon, Raksha Rank, and Etienne Chenevert. Fighting Fires from Space: Leveraging Vision Transformers for Enhanced Wildfire Detection and Characterization, April 2025. URL http://arxiv.org/abs/2504.13776. arXiv:2504.13776 [cs].

[59] George L. James, Ryeim B. Ansaf, Sanaa S. Al Samahi, Rebecca D. Parker, Joshua M. Cutler, Rhode V. Gachette, and Bahaa I. Ansaf. An Efficient Wildfire Detection System for AI-Embedded Applications Using Satellite Imagery. *Fire*, 6(4):169, April 2023. ISSN 2571-6255. doi:10.3390/fire6040169. URL https://www.mdpi.com/2571-6255/6/4/169. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[60] Gabriele Meoni, Roberto Del Prete, Lucia Ancos-Villa, Enrique Albalate-Prieto, David Rijlaarsdam, Jose Luis Espinosa-Aranda, Nicolas Longépé, Maria Daniela Graziano, and Alfredo Renga. E2E: Onboard satellite real-time classification of thermal hotspots events on optical raw data. *Astrodynamics*, May 2025. ISSN 2522-008X, 2522-0098. doi:10.1007/s42064-024-0249-x. URL https://link.springer.com/10.1007/s42064-024-0249-x.

[61] A.G. Davies, S. Chien, V. Baker, T. Doggett, J. Dohm, R. Greeley, F. Ip, R. Castanˇo, B. Cichy, G. Rabideau, D. Tran, and R. Sherwood. Monitoring active volcanism with

the Autonomous Sciencecraft Experiment on EO-1. *Remote Sensing of Environment*, 101(4):427–446, April 2006. ISSN 00344257. doi:10.1016/j.rse.2005.08.007. URL https://linkinghub.elsevier.com/retrieve/pii/S0034425705002543.

[62] Georges Labreche, David Evans, Dominik Marszk, Tom Mladenov, Vasundhara Shiradhonkar, Tanguy Soto, and Vladimir Zelenevskiy. OPS-SAT Spacecraft Autonomy with TensorFlow Lite, Unsupervised Learning, and Online Machine Learning. *IEEE Aerospace Conference*, March 2022. doi:10.1109/aero53065.2022.9843402. MAG ID: 4292231571 S2ID: 9e271e7e98c112431de73405b8888d3e02f1746c.

[63] Alessandro Marin, César Coelho, Florian Deconinck, Irina Babkina, Nicolas Longépé, and Massimiliano Pastena. *Phi-Sat-2: Onboard AI Apps for Earth Observation*. September 2021.

[64] Gianluca Giuffrida, Luca Fanucci, Gabriele Meoni, Matej Batic, Leonie Buckley, Aubrey Dunne, Chris Van Dijk, Marco Esposito, John Hefele, Nathan Vercruyssen, Gianluca Furano, Massimiliano Pastena, and Josef Aschbacher. The -Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–14, 2022. ISSN 0196-2892, 1558-0644. doi:10.1109/TGRS.2021.3125567. URL https://ieeexplore.ieee.org/document/9600851/.

[65] Håvard F. Grip, Johnny Lam, David S. Bayard, Dylan T. Conway, Gurkirpal Singh, Roland Brockers, Jeff H. Delaune, Larry H. Matthies, Carlos Malpica, Travis L. Brown, Abhinandan Jain, Alejandro M. San Martin, and Gene B. Merewether. Flight Control System for NASA's Mars Helicopter. In *AIAA Scitech 2019 Forum*, San Diego, California, January 2019. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-578-4. doi:10.2514/6.2019-1289. URL https://arc.aiaa.org/doi/10.2514/6.2019-1289.

[66] 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. ISSN 1542-9423. doi:10.2514/1.12923. URL https://arc.aiaa.org/doi/10.2514/1.12923.

[67] Avneep Dhingra. Building a real-time, on-demand solution for the 21st century — Payam Banazadeh, Capella Space, July 2019. URL https://geospatialworld.net/article/on-demand-solution-for-the-21st-century-capella-space/.

[68] Dennis Ogbe, Andre Jongeling, Zaid Towfic, Saba Janamian, David Foor, Bridget Wiley, Daniel Cho, Edwin Grigorian, Gregory Miles, Clayton Okino, Timothy Canham, Douglas Sheldon, and Joseph Sauvageau. The JPL Snapdragon Co-Processor: A Compact High-Performance Computer for Spaceflight Applications. In *2025 IEEE Aerospace Conference*, pages 1–12, March 2025. doi:10.1109/AERO63441.2025.11068431. URL https://ieeexplore.ieee.org/document/11068431/. ISSN: 2996-2358.

[69] David Rijlaarsdam, Tom Hendrix, Pablo T Toledano González, Alberto Velasco-Mata, Léonie Buckley, Juan Puig Miquel, Oriol Aragon Casaled, and Aubrey Dunne. The Next Era for Earth Observation Spacecraft: An Overview of CogniSAT-6, February 2024. URL https://www.techrxiv.org/users/746922/articles/718753-the-next-era-for-earth-observation-spacecraft-an-overview-of-cognisat-6?commit=8c953fb3boefd72daf3e69eabd23c63b845d1698.

[70] David Rijlaarsdam, Tom Hendrix, Pablo T Toledano Gonzalez, Alberto Velasco-Mata, Leonie Buckley, Puig Miquel, Oriol Aragon Casaled, and Aubrey Dunne. Autonomous Operational Scheduling on Cognisat-6 Based on Onboard Artificial Intelligence. 2024.

[71] J.S. Pearlman, P.S. Barry, C.C. Segal, J. Shepanski, D. Beiso, and S.L. Carman. Hyperion, a space-based imaging spectrometer. *IEEE Transactions on Geoscience and Remote Sensing*, 41(6):1160–1173, June 2003. ISSN 1558-0644. doi:10.1109/TGRS.2003.815018. URL https://ieeexplore.ieee.org/document/1220223/.

[72] Kathie Blackman and Jeff D'Agostino. A Consolidated ACS Flight Software Development Approach for the Earth Observing-1 Spacecraft. 1999.

[73] Kiri L Wagstaff, Alphan Altinok, Steve A Chien, Umaa Rebbapragada, Steve R Schaffer, David R Thompson, and Daniel Q Tran. Cloud Filtering and Novelty Detection using Onboard Machine Learning for the EO-1 Spacecraft. *International Joint Conference on Artificial Intelligence*, page 4, 2017.

[74] Sandra Hayden, Adam Sweet, and Scott Christa. Livingstone Model-Based Diagnosis of Earth Observing One. In *AIAA 1st Intelligent Systems Technical Conference*, Chicago, Illinois, September 2004. American Institute of Aeronautics and Astronautics. doi:10.2514/6.2004-6225. URL https://arc.aiaa.org/doi/10.2514/6.2004-6225.

[75] Rebecca Castano, Dominic Mazzoni, Nghia Tang, Ron Greeley, Thomas Doggett, Ben Cichy, Steve Chien, and Ashley Davies. Onboard classifiers for science event detection on a remote sensing spacecraft. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 845–851, Philadelphia PA USA, August 2006. ACM. ISBN 978-1-59593-339-3. doi:10.1145/1150402.1150519. URL https://dl.acm.org/doi/10.1145/1150402.1150519.

[76] Steve Chien, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Richard Castaño, Ashley Davies, Daniel Mandl, Stuart Frye, Bruce Trout, Jeff D'Agostino, Seth Shulman, Darrell Boyer, Sandra Hayden, Adam Sweet, and Scott Christa. *Lessons learned from autonomous sciencecraft experiment*. July 2005. doi:10.1145/1082473.1082798. Journal Abbreviation: Proceedings of the International Conference on Autonomous Agents Pages: 18 Publication Title: Proceedings of the International Conference on Autonomous Agents.

[77] Damhnait F. Gleeson, Robert T. Pappalardo, Steven E. Grasby, Mark S. Anderson, Benoit Beauchamp, Rebecca Castaño, Steve A. Chien, Thomas Doggett, Lukas Mandrake, and Kiri L. Wagstaff. Characterization of a sulfur-rich Arctic spring site and field

analog to Europa using hyperspectral data. *Remote Sensing of Environment*, 114 (6):1297–1311, June 2010. ISSN 00344257. doi:10.1016/j.rse.2010.01.011. URL https://linkinghub.elsevier.com/retrieve/pii/S0034425710000337.

[78] Lukas Mandrake, Umaa Rebbapragada, Kiri L. Wagstaff, David Thompson, Steve Chien, Daniel Tran, Robert T. Pappalardo, Damhnait Gleeson, and Rebecca Castaño. Surface Sulfur Detection via Remote Sensing and Onboard Classification. *ACM Transactions on Intelligent Systems and Technology*, 3(4):1–20, September 2012. ISSN 2157-6904, 2157-6912. doi:10.1145/2337542.2337562. URL https://dl.acm.org/doi/10.1145/2337542.2337562.

[79] David R. Thompson, Benjamin J. Bornstein, Steve A. Chien, Steven Schaffer, Daniel Tran, Brian D. Bue, Rebecca Castaño, Damhnait F. Gleeson, and Aaron Noell. Autonomous Spectral Discovery and Mapping Onboard the EO-1 Spacecraft. *IEEE Transactions on Geoscience and Remote Sensing*, 51(6):3567–3579, June 2013. ISSN 1558-0644. doi:10.1109/TGRS.2012.2226040. URL https://ieeexplore.ieee.org/document/6392936/.

[80] Steve Chien, Joshua Doubleday, David R. Thompson, Kiri L. Wagstaff, John Bellardo, Craig Francis, Eric Baumgarten, Austin Williams, Edmund Yee, Eric Stanton, and Jordi Piug-Suari. Onboard Autonomy on the Intelligent Payload Experiment CubeSat Mission. *Journal of Aerospace Information Systems*, 14(6):307–315, June 2017. ISSN 2327-3097. doi:10.2514/1.I010386. URL https://arc.aiaa.org/doi/10.2514/1.I010386.

[81] Ryoichi Imasu, Tsuneo Matsunaga, Masakatsu Nakajima, Yukio Yoshida, Kei Shiomi, Isamu Morino, Naoko Saitoh, Yosuke Niwa, Yu Someya, Yu Oishi, Makiko Hashimoto, Hibiki Noda, Kouki Hikosaka, Osamu Uchino, Shamil Maksyutov, Hiroshi Takagi, Haruma Ishida, Takashi Y. Nakajima, Teruyuki Nakajima, and Chong Shi. Greenhouse gases Observing SATellite 2 (GOSAT-2): mission overview. *Progress in Earth and Planetary Science*, 10(1):33, July 2023. ISSN 2197-4284. doi:10.1186/s40645-023-00562-2. URL https://progearthplanetsci.springeropen.com/articles/10.1186/s40645-023-00562-2.

[82] Hiroshi Suto, Fumie Kataoka, Nobuhiro Kikuchi, Robert O. Knuteson, Andre Butz, Markus Haun, Henry Buijs, Kei Shiomi, Hiroko Imai, and Akihiko Kuze. Thermal and near-infrared sensor for carbon observation Fourier transform spectrometer-2 (TANSO-FTS-2) on the Greenhouse gases Observing SATellite-2 (GOSAT-2) during its first year in orbit. *Atmospheric Measurement Techniques*, 14(3):2013–2039, March 2021. ISSN 1867-1381. doi:10.5194/amt-14-2013-2021. URL https://amt.copernicus.org/articles/14/2013/2021/. Publisher: Copernicus GmbH.

[83] Chloe Byrne. How NASA Is Testing AI to Make Earth-Observing Satellites Smarter – Ubotica Technologies Dynamic Targeting, August 2025. URL https://ubotica.com/how-nasa-is-testing-ai-to-make-earth-observing-satellites-smarter/.

[84] Alberto Candela, Jason Swope, Steve Chien, Hui Su, and Peyman Tavallali. Dynamic Targeting for Improved Tracking of Storm Features. In *IGARSS 2022 -*

2022 IEEE International Geoscience and Remote Sensing Symposium, pages 5313–5316, Kuala Lumpur, Malaysia, July 2022. IEEE. ISBN 978-1-6654-2792-0. doi:10.1109/IGARSS46834.2022.9883696. URL https://ieeexplore.ieee.org/document/9883696/.

[85] OPS-SAT Space Lab, . URL https://opssat.esa.int/ops-sat-1/.

[86] Georges Labrèche and Cesar Guzman Alvarez. SaaSyML: Software as a Service for Machine Learning On-board the OPS-SAT Spacecraft. In *2023 IEEE Aerospace Conference*, pages 1–9, March 2023. doi:10.1109/AERO55745.2023.10115531. URL https://ieeexplore.ieee.org/abstract/document/10115531. ISSN: 1095-323X.

[87] Juan Delfa Victoria, Alberto Candela, and Steve A Chien. Enhanced Dynamic Targeting for the OPS-SAT Cubesat. *International Workshop on Planning & Scheduling for Space (IWPSS)*, 2023.

[88] Landsat 8 | Landsat Science, November 2021. URL https://landsat.gsfc.nasa.gov/satellites/landsat-8/.

[89] Jacqueline Le Moigne, Michael M. Little, and Marjorie C. Cole. New Observing Strategy (NOS) for Future Earth Science Missions. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 5285–5288, Yokohama, Japan, July 2019. IEEE. ISBN 978-1-5386-9154-0. doi:10.1109/IGARSS.2019.8898096. URL https://ieeexplore.ieee.org/document/8898096/.

[90] Steve Chien. Integrated AI in Space: The Autonomous Sciencecraft on Earth Observing One. *Proceedings of the National Conference on Artificial Intelligence*, 2006.

[91] Steve Chien, Russell Knight, Steve Schaffer, David R Thompson, Brian Bue, and Martina Troesch. An Onboard Autonomous Response Prototype for an Earth Observing Spacecraft. *International Joint Conference on Artificial Intelligence Workshop on Artificial Intelligence in Space (AI Space, IJCAI 2015)*, 2015.

[92] Steve Chien and Martina Troesch. Heuristic Onboard Pointing Re-scheduling for an Earth Observing Spacecraft.

[93] Grégory Beaumet, Gérard Verfaillie, and Marie-Claire Charmeau. Feasibility of Autonomous Decision Making on Board an Agile Earth-Observing Salite. *Computational Intelligence*, 27(1):123–139, 2011. ISSN 1467-8640. doi:10.1111/j.1467-8640.2010.00375.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8640.2010.00375.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8640.2010.00375.x.

[94] Sreeja Nag, Alan Li, Alan S. Li, Alan S. Li, Vinay Ravindra, Vinay Ravindra, Vinay Ravindra, Marc Sanchez Net, Marc Sanchez Net, Kar-Ming Cheung, Rod Lammers, and Brian P. Bledsoe. Autonomous Scheduling of Agile Spacecraft Constellations with Delay Tolerant Networking for Reactive Imaging. April 2019. MAG ID: 2998771969.

[95] 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. ISSN 1940-3151, 2327-3097. doi:10.2514/1.I011215. URL https://arc.aiaa.org/doi/10.2514/1.I011215.

[96] Zaki Hasnain, James C Mason, Jason Swope, Joshua Vander Hook, and Steve Chien. Agile Spacecraft Imaging Algorithm Comparison for Earth Science. *International Workshop on Planning and Scheduling for Space (IWPSS)*, 2021.

[97] 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. ISSN 1940-3151, 2327-3097. doi:10.2514/1.I011233. URL https://arc.aiaa.org/doi/10.2514/1.I011233.

[98] Jason Swope, Steve Chien, Xavier Bosch-Lluis, Qing Yue, Peyman Tavallali, Mehmet Ogut, Isaac Ramos, Pekka Kangaslahti, William Deal, and Caitlyn Cooke. Using Intelligent Targeting to increase the science return of a Smart Ice Storm Hunting Radar. *International Workshop on Planning & Scheduling for Space (IWPSS)*, 2021.

[99] Violet Felt, Shreeyam Kacker, Joe Kusters, John Pendergrast, and Kerri Cahoy. Fast Ocean Front Detection Using Deep Learning Edge Detection Models. *IEEE Transactions on Geoscience and Remote Sensing*, 2023. Publisher: IEEE.

[100] Shreeyam Kacker, Alex Meredith, Joe Kusters, Hannah Tomio, Kerri Cahoy, and Violet Felt. On-orbit rule-based and deep learning image segmentation strategies. *AIAA SCITECH 2022 Forum*, January 2022. doi:10.2514/6.2022-0646. MAG ID: 4205910275 S2ID: 6647e3fa789a96d068eb556b2d346155a0b0eb52.

[101] A. Maillard, G. Verfaillie, C. Pralet, J. Jaubert, I. Sebbag, F. Fontanari, and J. Lhermitte. Adaptable Data Download Schedules for Agile Earth-Observing Satellites. *Journal of Aerospace Information Systems*, 13(3):280–300, March 2016. ISSN 1940-3151, 2327-3097. doi:10.2514/1.I010383. URL https://arc.aiaa.org/doi/10.2514/1.I010383.

[102] Adrien Maillard, Steve Chien, and Christopher Wells. Planning the Coverage of Solar System Bodies Under Geometric Constraints. *Journal of Aerospace Information Systems*, 18(5):289–306, May 2021. ISSN 2327-3097. doi:10.2514/1.I010896. URL https://arc.aiaa.org/doi/10.2514/1.I010896.

[103] S. Chien, B. Cichy, A. Davies, D. Tran, G. Rabideau, R. Castano, R. Sherwood, D. Mandl, S. Frye, S. Shulman, J. Jones, and S. Grosvenor. An autonomous earth-observing sensorWeb. *IEEE Intelligent Systems*, 20(3):16–24, May 2005. ISSN 1941-1294. doi:10.1109/MIS.2005.40. URL https://ieeexplore.ieee.org/abstract/document/1439475. Conference Name: IEEE Intelligent Systems.

[104] Steve A. Chien, Ashley G. Davies, Joshua Doubleday, Daniel Q. Tran, David Mclaren, Wayne Chi, and Adrien Maillard. Automated Volcano Monitoring Using Multiple Space and Ground Sensors. *Journal of Aerospace Information Systems*, 17(4):214–228, April

2020. ISSN 2327-3097. doi:10.2514/1.I010798. URL https://arc.aiaa.org/doi/10.2514/1.I010798.

[105] 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. ISSN 2327-3097. doi:10.2514/1.I010672. URL https://arc.aiaa.org/doi/10.2514/1.I010672.

[106] Mary Dahl, Juliana Chew, and Kerri Cahoy. Optimization of SmallSat Constellations and Low Cost Hardware to Utilize Onboard Planning. *AIAA ASCEND*, 2021. doi:10.2514/6.2021-4172. URL https://arc.aiaa.org/doi/abs/10.2514/6.2021-4172.

[107] Andrew Kitrell Kennedy. *Resource optimization algorithms for an automated coordinated CubeSat constellation*. Thesis, Massachusetts Institute of Technology, 2015. URL https://dspace.mit.edu/handle/1721.1/101497. Accepted: 2016-03-03T21:04:55Z.

[108] Shreeyam Kacker, Steve Chien, Kiruthika Devaraj, Isıl Demir, and Kerri Cahoy. Leveraging Realtime Meteorological Data for Dynamic Tasking of Agile Earth-Observing Satellites. 2025.

[109] Federico D'Anzeo, Thomas Dell, Mattia Rodegari, Steve Chien, Alberto Candela, Qing Yue, Marcin Kurowski, and Itai Zilberstein. Passive interception, decoding and processing of public broadcast L-Band EO data in LEO. In *Proceedings of the 5th ESA Workshop on Advanced Flexible Telecom Payloads*, ESA/ECSAT Harwell Campus, Didcot, United Kingdom.

[110] NOAA. GOES-R Series Data Book. URL https://www.goes-r.gov/downloads/resources/documents/GOES-RSeriesDataBook.pdf.

[111] 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. ISSN 0022-4650, 1533-6794. doi:10.2514/1.A34931. URL https://arc.aiaa.org/doi/10.2514/1.A34931.

[112] Diogo V. Andrade, Mauricio G. C. Resende, and Renato F. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, August 2012. ISSN 1381-1231, 1572-9397. doi:10.1007/s10732-012-9196-4. URL http://link.springer.com/10.1007/s10732-012-9196-4.

[113] Mingmin Zhao, Peder Olsen, and Ranveer Chandra. Seeing Through Clouds in Satellite Images. *IEEE Transactions on Geoscience and Remote Sensing*, 61:1–16, 2023. ISSN 1558-0644. doi:10.1109/TGRS.2023.3239592. URL https://ieeexplore.ieee.org/document/10025762/.

[114] Nicola Bianchessi and Giovanni Righini. Planning and scheduling algorithms for the COSMO-SkyMed constellation. *Aerospace Science and Technology*, 12(7):535–544, October 2008. ISSN 1270-9638. doi:10.1016/j.ast.2008.01.001. URL https://www.sciencedirect.com/science/article/pii/S1270963808000187.

[115] 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. ISBN 978-1-7281-2734-7. doi:10.1109/AERO47225.2020.9172258. URL https://ieeexplore.ieee.org/document/9172258/.

[116] World Cities Database | Simplemaps.com, . URL https://simplemaps.com/data/world-cities.

[117] Hi-Res SkySat Imagery Now Available, . URL https://www.planet.com/pulse/hi-res-skysat-imagery-available-via-planet-api/.

[118] Yingjie Xu, Xiaolu Liu, Renjie He, and Yingguo Chen. Multi-satellite scheduling framework and algorithm for very large area observation. *Acta Astronautica*, 167: 93–107, February 2020. ISSN 00945765. doi:10.1016/j.actaastro.2019.10.041. URL https://linkinghub.elsevier.com/retrieve/pii/S0094576519313694.

[119] James Richard Wertz, David F. Everett, and Jeffery John Puschell. *Space Mission Engineering: The New SMAD*. Microcosm Press, 2011. ISBN 978-1-881883-16-6. Google-Books-ID: 4JM7tAEACAAJ.

[120] T. Krauß. EXTRACTION OF CLOUD HEIGHTS FROM SENTINEL-2 MULTISPECTRAL IMAGES. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-1-2021:17–23, June 2021. ISSN 2194-9050. doi:10.5194/isprs-annals-V-1-2021-17-2021. URL https://isprs-annals.copernicus.org/articles/V-1-2021/17/2021/.

[121] SPOT6 User Guide, . URL https://www.spaceoffice.nl/blobs/Dataportaal/User_Guide_SPOT6_V1.0.pdf.

[122] Maxar Technologies. WorldView-3 Datasheet, . URL https://www.spaceimagingme.com/downloads/sensors/datasheets/DG_WorldView3_DS_2014.pdf.

[123] Olivier Amram. Pléiades Imagery User Guide. URL https://content.satimagingcorp.com/media/pdf/User_Guide_Pleiades.pdf.

[124] GeoEye-1 (OrbView-5) - eoPortal, . URL https://www.eoportal.org/satellite-missions/geoeye-1.

[125] James Lutes and Jacek Grodecki. Error Propagation in Ikonos Mapping Blocks. *Photogrammetric Engineering & Remote Sensing*, 70(8):947–955, August 2004. ISSN 00991112. doi:10.14358/PERS.70.8.947. URL http://openurl.ingenta.com/content/xref?genre=article&issn=0099-1112&volume=70&issue=8&spage=947.

[126] Pléiades Neo - Earth Online, . URL https://earth.esa.int/eogateway/missions/pleiades-neo.

[127] Maxar Technologies. WorldView-1 Product Quick Reference Guide, .

[128] Morton I. Kamien and Nancy L. Schwartz. *Dynamic Optimization, Second Edition: The Calculus of Variations and Optimal Control in Economics and Management*. Courier Corporation, April 2013. ISBN 978-0-486-31028-2. Google-Books-ID: liLCAgAAQBAJ.

[129] Xavier Bosch-Lluis, Pekka Kangaslahti, Isaac Ramos, Mehmet Ogut, Alan Tanner, Joelle Cooperrider, Joan Francesc Munoz-Martini, Qing Yue, William Deal, and Caitlyn Cooke. Smart Ice Cloud Sensing (SMICES): An Overview of its Submillimeter Wave Radiometer. In *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, pages 4296–4299, July 2022. doi:10.1109/IGARSS46834.2022.9883671. ISSN: 2153-7003.

[130] Adam Herrmann and Hanspeter Schaub. A Comparison of Deep Reinforcement Learning Algorithms for Earth-Observing Satellite Scheduling. *AAS/AIAA Spaceflight Mechanics Meeting*, 2023.

[131] Nicola Bianchessi, Jean-François Cordeau, Jacques Desrosiers, Gilbert Laporte, and Vincent Raymond. A heuristic for the multi-satellite, multi-orbit and multi-user management of Earth observation satellites. *European Journal of Operational Research*, 177(2):750–762, March 2007. ISSN 03772217. doi:10.1016/j.ejor.2005.12.026. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221706000051.

[132] Adam P. Herrmann and Hanspeter Schaub. Monte Carlo Tree Search Methods for the Earth-Observing Satellite Scheduling Problem. *Journal of Aerospace Information Systems*, 19(1):70–82, January 2022. ISSN 2327-3097. doi:10.2514/1.I010992. URL https://arc.aiaa.org/doi/10.2514/1.I010992.

[133] Sean Augenstein, Alejandra Estanislao, Emmanuel Guere, and Sean Blaes. Optimal scheduling of a constellation of earth-imaging satellites, for maximal data throughput and efficient human management. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, pages 345–352, 2016. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13784.

[134] E. Bensana, M. Lemaitre, and G. Verfaillie. Earth Observation Satellite Management. *Constraints*, 4(3):293–299, September 1999. ISSN 1572-9354. doi:10.1023/A:1026488509554. URL https://doi.org/10.1023/A:1026488509554.

[135] Hong-Jen Chen and Brij Agrawal. Method of Slewing the Spacecraft to Minimize Settling Time. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Monterey, California, August 2002. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-108-3. doi:10.2514/6.2002-4656. URL https://arc.aiaa.org/doi/10.2514/6.2002-4656.

[136] William J. Wolfe and Stephen E. Sorensen. Three Scheduling Algorithms Applied to the Earth Observing Systems Domain. *Management Science*, 46(1):148–166, January 2000. doi:10.1287/mnsc.46.1.148.15134. MAG ID: 2148027993.

[137] Sean Augenstein. Optimal Scheduling of Earth-Imaging Satellites with Human Collaboration via Directed Acyclic Graphs. *AAAI Spring Symposium Series*, 2014.

[138] 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. ISSN 02731177. doi:10.1016/j.asr.2017.11.010. URL https://linkinghub.elsevier.com/retrieve/pii/S0273117717308050.

[139] Adrien Hadj-Salah, Rémi Verdier, Clément Caron, Mathieu Picard, and Mikaël Capelle. Schedule Earth Observation satellites with Deep Reinforcement Learning, November 2019. URL http://arxiv.org/abs/1911.05696. arXiv:1911.05696 [cs].

[140] Adam Herrmann and Hanspeter Schaub. Reinforcement Learning for the Agile Earth-Observing Satellite Scheduling Problem. *IEEE Transactions on Aerospace and Electronic Systems*, pages 1–13, 2023. ISSN 0018-9251, 1557-9603, 2371-9877. doi:10.1109/TAES.2023.3251307. URL https://ieeexplore.ieee.org/document/10058020/.

[141] Xin Wang, Fanyu Zhao, Zhong Shi, and Zhonghe Jin. Deep Reinforcement Learning-Based Periodic Earth Observation Scheduling for Agile Satellite Constellation. *Journal of Aerospace Information Systems*, pages 1–12, May 2023. ISSN 1940-3151, 2327-3097. doi:10.2514/1.I011209. URL https://arc.aiaa.org/doi/10.2514/1.I011209.

[142] Ahmet B. Keha, Ketan Khowala, and John W. Fowler. Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, 56(1):357–367, February 2009. ISSN 03608352. doi:10.1016/j.cie.2008.06.008. URL https://linkinghub.elsevier.com/retrieve/pii/S0360835208001265.

[143] Bobby Glenn Holden. *Onboard distributed replanning for crosslinked small satellite constellations*. Thesis, Massachusetts Institute of Technology, 2019. URL https://dspace.mit.edu/handle/1721.1/122513. Accepted: 2019-10-11T21:59:43Z ISBN: 9781121262690.

[144] James Boerkoel, James Mason, Daniel Wang, Steve Chien, and Adrien Maillard. An Efficient Approach for Scheduling Imaging Tasks Across a Fleet of Satellites. *International Workshop on Planning \& Scheduling for Space (IWPSS)*, 2021.

[145] GOES Overview, . URL https://www.noaasis.noaa.gov/GOES/goes_overview.html.

[146] MSG services | EUMETSAT, May 2023. URL http://www.eumetsat.int/msg-services.

[147] Himawari-8/9 Leaflet, . URL https://www.jma.go.jp/jma/jma-eng/satellite/materials/Himawari89/himawari89_leaflet/201703_leaflet89.pdf.

[148] ABI Bands Quick Information Guides, . URL https://www.goes-r.gov/mission/ABI-bands-quick-info.html.

[149] J Schmid. The SEVIRI Instrument. URL https://www-cdn.eumetsat.int/files/2020-04/pdf_ten_msg_seviri_instrument.pdf.

[150] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, February 2011. ISSN 0377-2217. doi:10.1016/j.ejor.2010.03.045. URL https://www.sciencedirect.com/science/article/pii/S0377221710002973.

[151] Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Running Errands in Time: Approximation Algorithms for Stochastic Orienteering. *Mathematics of Operations Research*, 40(1):56–79, February 2015. ISSN 0364-765X, 1526-5471. doi:10.1287/moor.2014.0656. URL https://pubsonline.informs.org/doi/10.1287/moor.2014.0656.

[152] Irina Dolinskaya, Zhenyu (Edwin) Shi, and Karen Smilowitz. Adaptive orienteering problem with stochastic travel times. *Transportation Research Part E: Logistics and Transportation Review*, 109:1–19, January 2018. ISSN 13665545. doi:10.1016/j.tre.2017.10.013. URL https://linkinghub.elsevier.com/retrieve/pii/S1366554516300242.

[153] Antonio Torralba, Phillip Isola, and William T. Freeman. *Foundations of Computer Vision*. MIT Press, April 2024. ISBN 978-0-262-04897-2. Google-Books-ID: xtC_EAAAQBAJ.

[154] Herbert A. David and Haikady N. Nagaraja. *Order Statistics*. John Wiley & Sons, March 2004. ISBN 978-0-471-65401-8. Google-Books-ID: bdhzFXg6xFkC.

[155] William Feller. *An Introduction to Probability Theory and Its Applications, Volume 2*. John Wiley & Sons, January 1991. ISBN 978-0-471-25709-7. Google-Books-ID: rxadEAAAQBAJ.

[156] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *arXiv preprint arXiv:1812.05905*, 2018.

[157] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, January 2022. URL http://arxiv.org/abs/2201.02177. arXiv:2201.02177 [cs].

[158] Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J Michaud, Max Tegmark, and Mike Williams. Towards Understanding Grokking: An Effective Theory of Representation Learning. 2022.

[159] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s, March 2022. URL http://arxiv.org/abs/2201.03545. arXiv:2201.03545 [cs].

[160] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi:10.1038/nature16961. URL https://www.nature.com/articles/nature16961. Publisher: Nature Publishing Group.