75th International Astronautical Congress (IAC), Milan, Italy, 14-18 October 2024.

IAC-24-D1.2.4

BSK-RL: Modular, High-Fidelity Reinforcement Learning Environments for Spacecraft Tasking

Mark A. Stephenson*, Hanspeter Schaub†
Ann and H.J. Smead Department of Aerospace Engineering Sciences, 431 UCB, Colorado Center for Astrodynamics Research, Boulder, CO, 80309.

Reinforcement learning (RL) is a highly adaptable framework for generating autonomous agents across a wide domain of problems. While RL has been successfully applied to highly complex, real-world systems, a significant amount of the literature studies abstractions and idealized versions of problems. This is especially the case for the field of spacecraft tasking, in which even traditional preplanning approaches tend to use highly simplified models of spacecraft dynamics and operations. When simplified methods are tested in a full-fidelity simulation, they often lead to conservative solutions that are suboptimal or aggressive solutions that are infeasible. As a result, there is a need for a high-fidelity spacecraft simulation environment to evaluate RL-based and other tasking algorithms. This paper introduces BSK-RL, an open-source Python package for creating and customizing reinforcement learning environments for spacecraft tasking problems. It combines Basilisk — a high-speed and high-fidelity spacecraft simulation framework — with abstractions of satellite tasks and operational objectives within the standard Gymnasium API wrapper for RL environments. The package is designed to meet the needs of RL and spacecraft operations researchers: Environment parameters are easily reproducible, customizable, and randomizable. Environments are highly modular: satellite state and action spaces can be specified, mission objectives and rewards can be defined, and the satellite dynamics and flight software can be configured, implicitly introducing operational limitations and safety constraints. Heterogeneous multi-agent environments can be created for more complex mission scenarios that consider communication and collaboration. Training and deployment using the package are demonstrated for an Earth-observing satellite with resource constraints.

Keywords: reinforcement learning, autonomy, satellite constellations, spacecraft tasking, open-source software

Contents

1. Introduction

Autonomous spacecraft planning and scheduling has become of increasing interest as the number of satellites and complexity of mission architectures have grown in response to easier access to space [1]. For large constellations, ground-based, operator-driven planning is expensive or infeasible due to the number of satellites, while onboard distributed autonomy provides robustness to any single point of failure. In missions where the objective changes rapidly or the ability to communicate with the satellites is limited, the ability to adapt onboard to new information is crucial for maximizing performance.

Many of the methods commonly used for spacecraft scheduling have two key limitations when compared to distributed, autonomous systems: 1) They use open-loop, ground-based planners, with online plan correction limited by solver complexity and spacecraft hardware constraints; and 2) they plan over expert-designed abstractions of the problem space [2]. Such planners for task scheduling use a variety of common combinatorial optimization algorithms, such as mixed-integer linear program (MILP), iterative local search (ILS), constraint programming, and genetic methods [3–6]. Generally, these methods work by generating a representation of the problem as a graph [7] and/or a combination of binary and continuous constrained variables based on an expert understanding of the problem. The size of the optimization problem grows combinatorially with the number of spacecraft and tasks, leading to problems that can be computationally prohibitive to solve, or that result in suboptimal solutions due to a limited solution time. The resulting plans are not responsive to the actual system dynamics, so unexpected performance or opportunistic events may depress the performance of the plan. In some cases, uncertainty is accounted for in the deterministic plan [8, 9]. In other cases, replanning over a short horizon is necessary, but the cost of common solvers can easily exceed onboard capabilities [10].

Reinforcement learning (RL) offers a promising alternative that addresses these challenges to generate autonomous agents. Agents learn in a simulation of the satellite tasking environment, allowing for arbitrarily complex models of the system's dynamics, constraints, and objectives to be maximized over. The resulting policy is a closed-loop, computationally low-cost system that can be deployed on the spacecraft, allowing it to react in real-time to new information or unexpected events (e.g. faults or opportunistic events) that it encounters. Research applying RL to spacecraft control and tasking problems has proliferated in the past five years. The first formulations of tasking as varieties of Markov decision processes (MDPs) appear by Harris [11], Eddy [12], and Hadj-Salah [13] applied to Earth-observing scheduling problems. This domain has expanded into algorithms that are fine-tuned for particular problem statements [14, 15], challenges associated with applying RL in a more flight-like scenario [16, 17], and various treatments of the multiagent scheduling problem [18–21]. Relative motion control for inspection of small bodies [22, 23] and rendezvous and docking [24, 25] have emerged as popular continuous control applications of RL in the space domain. Even among these examples, many use highly simplified models of the system dynamics for task transitions, task completion, and resource constraints; as a result, they do not fully leverage the capabilities of RL to work on arbitrarily complex environments.

The standardization of RL environments has been of key importance for improving the quality, efficiency, and reproducibility of research in the field of reinforcement learning as a whole [26]. Widespread adoption of the Python-based Gymnasium (formerly Gym, developed by OpenAI) environment API and PettingZoo multiagent API has led to a de facto standard for interfacing RL algorithms with environments [27, 28]. Specific environments implemented using these APIs have also emerged as standard benchmarks for RL algorithms: Atari games like Breakout, Tetris, and Adventure have become key benchmarks for image-input, discrete control environments [29] [30]. The MuJoCo physics engine has been interfaced with Gymnasium to create a variety of continuous control environments like Ant, Hopper, and Half-Cheetah, as well as arbitrarily complex multibody physics-based environments [31]. Gymnasium interfaces have been developed for modern video games such as Minecraft [32] and Starcraft II [33], providing challenging benchmarks for RL algorithms in complex, high-dimensional, and partially observable environments. While these environments provide a diversity of open-source environments (or open-source interfaces into closed-source software) for RL research, they are primarily seen as a testing ground for algorithms; the environments themselves are not the focus of research. MuJoCo is an exception, as arbitrary physics-based environments of interest can be created within the engine and interfaced with Gymnasium.

Within the space domain, open-source RL environments are lacking. Other than BSK-RL, the only example identified in a search is "KSPDG: Kerbal Space Program Differential Games", which implements various non-cooperative spacecraft control tasks using the Gymnasium API and the video game Kerbal Space Program (KSP) as a physics engine [34]. This environment is designed primarily as a real-time evaluation environment of policies and control methods developed externally to the environment, encouraging users to treat KSPDG as "reality" when trying to overcome the sim-to-real gap. As a result, it is not appropriate for training agents due to the high computational overheads and real-time simulation of the KSP backend. While aforementioned spacecraft RL research has defined and implemented environments for particular problems, they tend to be low-fidelity, problem-specific models that lack a maintained, open-source repository for general use.

BSK-RL aims to fill this gap by providing a high-fidelity, high-speed, open-source, and modular environment for spacecraft tasking problems with an open-source repository[*] and user-friendly documentation[†]. At its core, BSK-RL combines Basilisk [35], a spacecraft dynamics simulation package, with abstract mission objectives, wrapped together in the Gymnasium API. The primary focus of the package is discrete tasking, but it can be extended to continuous control. In this work, the design and capabilities of BSK-RL are reviewed, and an example environment is presented.

2. Background

The two packages of open-source software that BSK-RL is built upon are Basilisk, a spacecraft simulation frame-

[*] `https://github.com/AVSLab/bsk_rl/`

[†] `https://avslab.github.io/bsk_rl/`

work, and Gymnasium, a standard API for reinforcement learning environments.

2.1. Basilisk

Basilisk[‡] is a modular spacecraft simulation framework written in C and C++ with a Python interface [35]. The package is capable of simulating spacecraft with features such as multibody dynamics (including reaction wheels and actuated solar arrays), environmental effects (such as planetary gravitational forces and atmospheric drag), power and data storage subsystems, and onboard flight software with actuator-level control. The architecture enables relatively complex configurations to achieve simulation speeds of about 500 to 1000× real time. This combination of speed and configurability makes Basilisk an attractive environment for reinforcement learning. Alternative commercial options suffer from having large computational overheads, limited APIs, and other drawbacks of closed-source software.

2.2. Gymnasium and PettingZoo

Gymnasium is the standard interface for defining MDPs in Python [27]. All major reinforcement learning libraries, such as RLlib [36], and Stable Baselines [37] are compatible with the Gymnasium API.

Environments implement two main functions: The reset function `obs, info = env.reset(seed)` sets up the initial state of an environment and handles environment condition randomization. The step function `obs, reward, term, trunc, info = env.step(action)` provides the fundamental means of interaction between an agent and its environment: The environment is updated depending on the action selected by the agent, and the new state and reward are returned (along with information about the episode's status).

To supply the environment interface to the learning agent, the environment implements the `observation_space` and `action_space` properties, respectively define the domains for the observations the agent expects to receive and the actions that it can take.

PettingZoo extends the Gymnasium API to handle cooperative and competitive multiagent scenarios [28]. In particular, BSK-RL adopts the PettingZoo Parallel API for the multiagent scenario `ConstellationTasking`.

3. Design

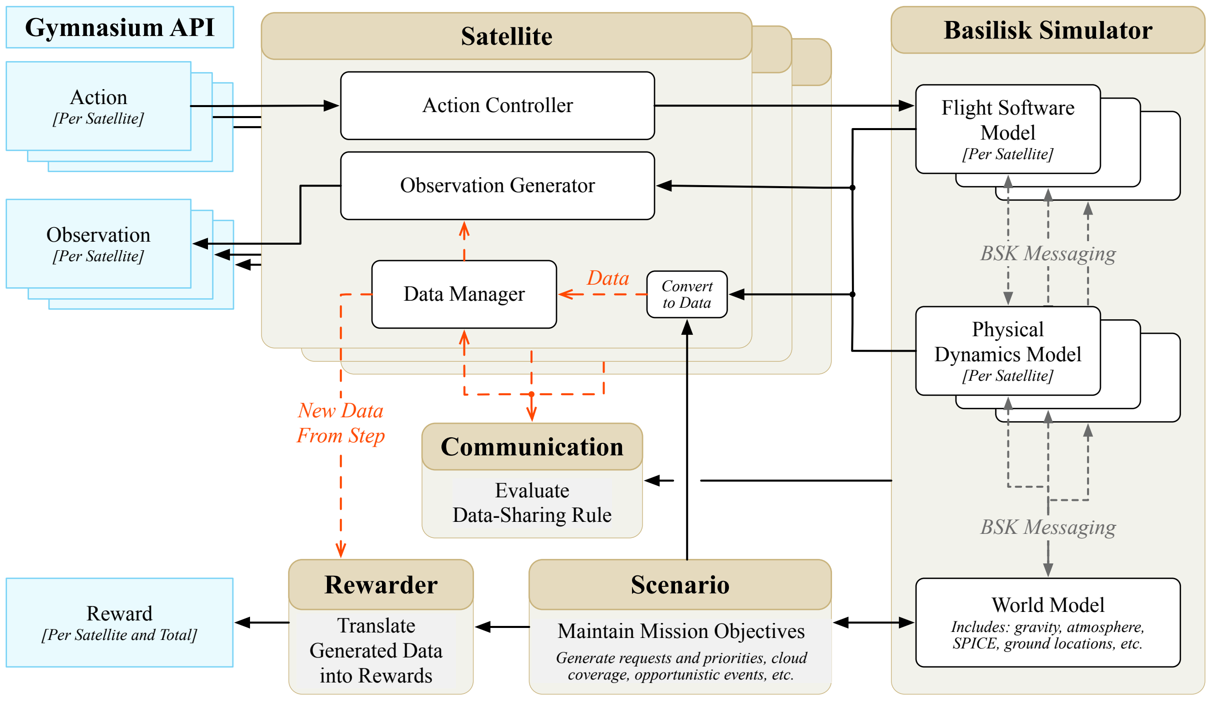
BSK-RL's architecture can be broadly categorized into three areas: 1) the underlying Basilisk simulation, which gives the physical behavior of the satellites in the environment; 2) the satellite agents that act in the environment, with configurable interfaces for observations and actions; 3) the optimization objective of the environment, which is specialized for data collection tasks but generalizes to other objectives. Working together, the environment connects the dynamics of the high-fidelity simulation to abstract tasks and goals that must be achieved. Finally, multiagent capabilities are built-in, allowing for complex, constellation-based scenarios to be created with communication between agents. Figure 1 shows the overall architecture of the BSK-RL environment.

3.1. Spacecraft Simulation

The Basilisk simulation is taken as the ground-truth for the physical behavior of the satellite as controlled by flight software modes in the simulation environment. The lifecycle of the simulator is as follows:

1) A new Basilisk simulation is constructed each time the environment is reset. First, the models — collections of Basilisk modules — used by each part of the simulator are identified. One is for the "world" that the satellites act in, containing simulations of gravity, the atmosphere, and solar position among other effects. Each satellite has two models associated with it: The dynamics module includes all of the physical components of the satellite, such as the bus, actuators, solar arrays and batteries, and data collection and transmission systems. The flight software model defines flight modes — sets of algorithms — that can be enabled and disabled on the satellite to command specific actuator behavior.
2) The models are initialized from a dictionary of simulation parameters. These values include mass properties, power draws, data rates, orbit parameters, epoch, and flight software algorithm settings, among others. A dictionary of default values can be easily generated from the classes that define the models. This allows the user to see how the simulation can be configured without *needing* to set all parameters. Parameters can be set directly by value or with functions that randomize the parameter on each reset. The latter is useful for randomizing initial conditions, such as the epoch and orbit in order to learn a policy that generalizes over domains. The appendix gives an example of a dictionary of satellite parameter overrides and randomizers.
3) When the environment is stepped, flight software modes are enabled and disabled depending on the action selected. In some cases, additional values in the flight software are changed to correspond to the selected action, such as setting the desired pointing direction to a particular target in the attitude controller. The translation between actions and flight software is discussed in the next section. Once the flight software is configured, the simulation is

[‡]`http://hanspeterschaub.info/basilisk/`



Gymnasium API
Action [Per Satellite]
Observation [Per Satellite]
Reward [Per Satellite and Total]
Satellite
Action Controller
Observation Generator
Data Manager
Data
Convert to Data
New Data From Step
Communication
Evaluate Data-Sharing Rule
Rewarder
Translate Generated Data into Rewards
Scenario
Maintain Mission Objectives
Generate requests and priorities, cloud coverage, opportunistic events, etc.
Basilisk Simulator
Flight Software Model [Per Satellite]
BSK Messaging
Physical Dynamics Model [Per Satellite]
BSK Messaging
World Model
Includes: gravity, atmosphere, SPICE, ground locations, etc.

Fig. 1 The architecture of the BSK-RL environment.

propagated until one of a few conditions is met. In many cases, these conditions are action-specific task durations or global maximum step durations. However, some actions that take an unpredictable time to complete may add additional checks for propagator termination. In light of the variable-duration environment steps, it is possible to view the resulting environment as a semi-MDP, which groups MDP substeps into multi-step actions or tasks [12, 38, 39].

4) Once the episode is complete, the simulator is deleted to prevent memory leaks.

Key to the realism of this environment is that the only interaction between the agent and the simulation is via the flight software, as would be the case on a real satellite.

Predefined models with custom parameters are generally sufficient for most environment configurations, but the user can define their own models by subclassing the provided models. This allows for the use of mission-specific flight software algorithms and satellite geometries.

3.1.1. Failure

The simulation models also check for failure conditions. In the single-agent environment, failure terminates the episode; in the multiagent environment, failure removes the failed satellite from the list of active agents. Optionally, a penalty can be subtracted from the step reward on failure.

Each dynamics and flight software model can define functions decorated with `@aliveness_checker` that will be evaluated at each step to ensure that the state of the Basilisk simulation is valid. As a result, arbitrary checkers can be easily defined. Commonly used checkers include the evaluation battery charge level, reaction wheel saturation, and orbital altitude.

3.2. Defining The RL Interface

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

```
class ScanningSatellite(Satellite):
    observation_spec = [SatProperties(...),
    ...]
    action_spec = [Charge(), Desat(), ...]
    dyn_model = MyDynamicsModel
    fsw_model = MyFSWModel
```

This class can then be instantiated to create new agents in the environment.

3.2.1. *Observation Specification*

The observation is specified as a list of observation objects. These can be configured to fetch values from the Basilisk simulation and other abstractions within the environment; normalization of these values — as is commonly necessary in deep reinforcement learning (DRL) — is also supported. The observation space is automatically inferred from the specification. An example observation specification is given:

```
observation_spec = [
obs.SatProperties(
    dict(prop="storage_level_fraction"),
    dict(prop="battery_charge_fraction"),
    dict(prop="wheel_speeds_fraction"),
    dict(prop="instrument_pointing_error",
    norm=np.pi),
    dict(prop="solar_pointing_error", norm=np.
    pi),
),
obs.OpportunityProperties(
    dict(prop="opportunity_open", norm=5700),
    dict(prop="opportunity_close", norm=5700),
    type="ground_station",
    n_ahead_observe=1,
),
obs.Eclipse(norm=5700),
obs.Time(),
]
```

In this specification, five properties (some scalar and some vector) are observed from the satellite and normalized. Information for each of the next 16 upcoming ground targets is also included. Information about the next eclipse time and overall episode time is also included.

A variety of useful observation types are defined in the package, including:

- `SatProperties`: Used to extract and normalize arbitrary properties from the satellite's flight software and dynamics models. Commonly desired properties are included in the default models, but additional arbitrary properties can be easily added by extending the model classes, as is demonstrated in the appendix.
- `OpportunityProperties`: Used to fetch information about the next *N* upcoming ground access opportunities of a particular type. These can include imaging targets or ground stations for downlink.
- `Eclipse`: The time until the start and end of the soonest (or current) eclipse.
- `Time`: The current time in the simulation, by default normalized by episode length. This can help with learning semi-MDPs.

The resulting observations can be returned in a flattened array or as a human-readable dictionary; this yields compatibility with most RL libraries while allowing for easy debugging and interpretation.

3.2.2. *Action Specification*

Similar to the observation, the actions are specified as a list of configurable action objects. The list of actions translates the policy's output (e.g. the index of a task) to the flight software settings that will be used to perform the task in simulation. Taking a discrete list of tasks as an example, the action specification could look as follows:

```
action_spec = [
    act.Scan(duration=180.0),
    act.Charge(duration=120.0),
    act.Downlink(duration=60.0),
    act.Desat(duration=60.0),
]
```

Many actions are defined to enable a certain mode for a set duration:

- `Charge`: Maneuvers the satellite to point its solar arrays at the sun to maximize the battery charge rate. Charging is ultimately dependent on the underlying simulation, so if the satellite is in eclipse or is maneuvering from a non-sun pointing attitude, this mode does not guarantee charging.
- `Desat`: Fire the reaction control thrusters to desaturate the reaction wheels. Since the reaction wheel power draw is a function of wheel speeds, desaturation is necessary to minimize power consumption or to prevent wheel saturation.
- `Downlink`: Enable the downlink transmitter and associated power costs. If within range of a ground station, data will be offloaded at a fixed baud rate, freeing space in the storage unit for new data.
- `Scan`: Point the instrument nadir and continuously collect data once the instrument is within some threshold of the desired direction. This action is used for continuous imaging tasks, where no choices are made about what to image.

For point-imaging scenarios, the `Image` action has more complex behavior. The action object generates configurable *N* actions that correspond to imaging each of the next *N* upcoming targets. When selected, the action sets the flight software mode to point the instrument at the target and collect an image once within range of the target and settled within some threshold. The simulation is then propagated until the target has been successfully imaged, until the opportunity has closed without imaging the target, or until some maximum step duration is reached. This prevents the satellite from having idle time once the task is complete (or can no longer be completed). This behavior can be overriden by disabling the satellite's `variable_interval` setting to make all steps propagate for the same duration.

Currently, only discrete tasking actions are implemented. However, the package is designed to allow for the definition of continuous action types for control problems.

3.3. Objective Abstractions

The final components of the environment define the mission objectives. The scenario represents what is of interest to the satellites, such as imaging requests and obstructions. The data collection system translates between the simulation state and data that contributes to the mission objective. The rewarder calculates the per-agent reward based on the scenario and the data collected by each agent.

3.3.1. Scenario

The scenario represents what is of interest to agents toward the mission objective. BSK-RL's current primary focus is Earth-observing tasks, so nadir scanning scenarios and point-target imaging scenarios are included in the package. The scenario could be extended to represent other objectives, such as inspection tasks or rendezvous and docking.

Many scenarios in BSK-RL are point imaging-based, which demonstrates a range of scenario capabilities. In these scenarios, a new list of requests is generated at the start of each episode, according to some distribution of locations and priorities. `UniformTargets` and `CityTargets` supply apriori lists of requests distributed uniformly over Earth or at the locations of randomly selected cities. The scenario could be used to model obstructions, such as cloud coverage, or targets that are not persistently available and change over time, such as wildfires.

3.3.2. Data Collection

The data collection system determines how each agent is contributing to the mission objective, by converting the simulation state and scenario information into units of data collected. Data does not necessarily correspond to a scientific/information product; it is a unit that represents effort towards the objective.

For example, consider an agile Earth-observing satellite (AEOS) scenario. If in the Basilisk simulation the amount of data in the storage unit buffer corresponding to a given request increases, the data manager identifies that the corresponding request defined in the scenario has been fulfilled and creates a unit of data to store with the satellite (to represent the satellite's knowledge of fulfillment) and to use in the reward calculation. See subsubsection 3.4.1 for a discussion on how this data can be shared with other agents.

3.3.3. Reward Allocation

Finally, the rewarder acts akin to a global critic for the environment. Given the per-agent new data at a given step and the current state of the environment, a reward is calculated for each agent. For example, in an AEOS scenario, the rewarder yields reward based on priority for each previously unfulfilled request fulfilled.

3.4. Multiagent Capabilities

A major motivation for this package is the proliferation of complex, multiagent space systems. Towards this end, multiagent scenarios are natively supported with `ConstellationTasking` and defined similarly to single-agent scenarios. This environment can be used either as a multiagent training environment or as a way of testing single-agent policies in a distributed system.

3.4.1. Communication Methods

In some scenarios, knowing what data has been collected by other agents is important for completing a goal. Transmitting data may prevent duplication of effort on already-complete tasks or inform other agents about tasks that require collaboration to satisfy. Towards this, a general notion of communication is implemented for multiagent scenarios.

Each communication method specifies which agents should share information at the end of each step. Of the implemented methods, `NoCommunication` allows for no data sharing, `FreeCommunication` allows every agent to share data with every other agent, and `LOSCommunication` and its variants allow for communication between satellites that had a clear line-of-sight during the previous step. The system is extendable to include more complex constraints on communication, such as requiring an explicit "communicate" action to trigger the data sharing step.

3.5. Formalized as a POMDP

It is beneficial to consider how the resulting environment maps to the partially-observable Markov decision process (POMDP) formalism. A POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R, O, Z)$, with the environment providing a generative model $G(s, a) = s'$:

- **State Space $\mathcal{S}$:** The state space is the combination of state spaces required for each component of the Basilisk simulation and environment. The space is impractically large to learn on directly. Many of the elements, such as internal integrator substates, are non-physical and not relevant to the tasking problem.
- **Observation Space O and Observation Probability Function Z:** The observation space is defined by the composition of satellite observation specifications and generally takes the form of a subset of the elements of the state space and transformations thereof (e.g. normalization, frame transformations, etc.). Generally, the observation is a deterministic

function of the state, though noise could be added in the observation specification to perform analysis similar to that in [40].
- **Action Space $\mathcal{A}$:** The action space is likewise defined by the composition of satellite action specifications.
- **Reward Function $R(s, a, s')$:** The reward function is arbitrary and defined by the rewarder.
- **Transition Probability Function $T(s'|s, a)$:** The transition function is deterministic and generated by the simulator, resulting in $T(G(s, a)|s, a) = 1$. The simulator is propagated until some condition is met to stop propagation, such as the completion of a task or a maximum step duration. It is valid to treat steps as single steps in a MDP where time is part of the state, or as the concatenation of substeps in a semi-MDP.

4. Results

A demonstration of BSK-RL is given for a single-satellite environment in which the satellite must maximize the amount of data collected in a nadir-pointing scanning mode while managing power and storage constraints. The environment is trained using RLlib's implementation of proximal policy optimization (PPO) [36, 41].

4.1. Demonstration Environment

The demonstration environment consists of a single satellite with four flight modes: nadir-pointing scanning, downlink, reaction wheel desaturation, and charging. The objective of the satellite is to collect as much data as possible in the nadir-pointing scanning mode while maintaining resources at proper levels. The reward function yields a reward proportional to the amount of data collected, normalized such that the maximum reward for continuous data collection over the entire episode is 1.0. A penalty of -1.0 is applied if the satellite dies (i.e. violates a resource constraint). The action space corresponds to the set of four flight modes; the observation space is a set of properties of the satellite and the environment, such as the battery charge level, the amount of data in the storage buffer, and the time until the next eclipse and downlink opportunity.

Data collection is subject to two conditions. First, the satellite's instrument must be pointing nadir within a 0.1 radian threshold to collect images. This is a constraint enforced by the flight software running within the Basilisk simulation. When the scanning mode is enabled, the satellite may need to slew to the nadir-pointing attitude, preventing immediate data collection. Second, there must be available space in the data buffer to collect images. This constraint is implicit to the Basilisk dynamics simulation, which does not allow the data buffer to overflow. Since the reward is generated from data collected, a full buffer prevents any reward from being yielded. The satellite must enter the downlink mode when over one of the seven default ground stations (Boulder, Merritt Island, Singapore, Weilheim, Santiago, Dongara, and Hawaii) to decrease the amount of data in the buffer and free space for new data.

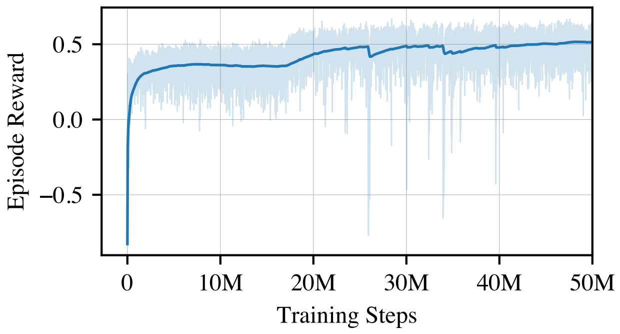


Fig. 2 Average episode returns over the course of training.

The satellite must also maintain power above zero for the full five-orbit episode. Power draws are implicit to the dynamics simulation and include baseline draws for bus operation, reaction wheel power draws as a function of wheel speeds, and mode-specific power draws for activating the instrument or transmitter. Charging always occurs based on the solar panel orientation and the eclipse status. However, the satellite can enter the charging flight mode to point the panels directly at the sun (which is ineffective if the satellite is in eclipse) and the reaction wheel desaturation mode to fire the thrusters to desaturate the reaction wheels, decrease the power draw due to high wheel speeds.

Code for the demonstration environment is given in the example code snippets throughout the paper and appendix, as well as in the online documentation[§].

4.2. Training Results

The environment was trained using PPO for 50M steps. Figure 2 shows the overall training curve: the cumulative reward returned by each episode over the course of training. There are two components to this curve: Is the satellite staying alive? and, how much data is the satellite collecting?

Regarding aliveness, Figure 3 shows that the satellite quickly learns that it must manage its power to survive for the full five orbits by desaturating reaction wheel speed and by entering a charging flight mode. By the end of training, 94.5% of the episodes are completed successfully. These failures represent a combination of cases: those with poor resource management and those where the satellite is initialized in a no-win situation (e.g.

[§]`https://avslab.github.io/bsk_rl/examples/rllib_training.html`

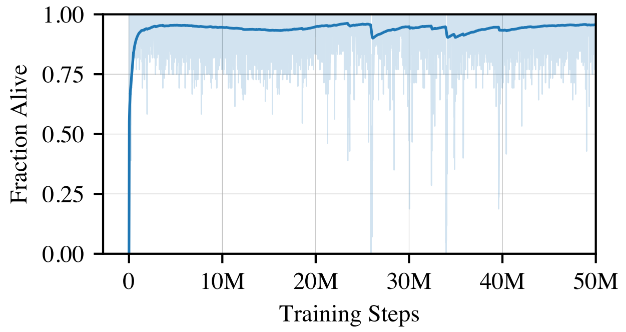


Fig. 3 Fraction of episodes in training where the agent survives for all five orbits.

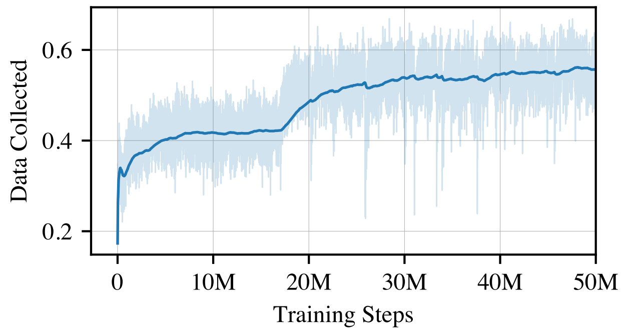


Fig. 4 Total data collected during episode; maximum 1.0 for continuous scanning.

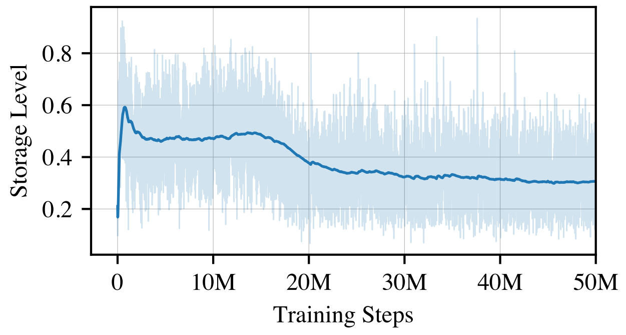


Fig. 5 Fraction of data buffer filled at end of episode.

low power during eclipse). Practically, shielding could be used to guarantee safe operation, but those methods are outside the scope of this paper [42, 43].

The satellite also successfully learns the task of data collection. Figure 4 shows that the satellite's policy adapts to collect more data over the course of training. Initially, it achieves this by increasing the time spent in the scanning mode while decreasing excess time spent in the charging mode. However, the satellite is also limited by its storage capacity. Figure 5 shows that at around 17M steps, the policy begins to better optimize downlink, with average storage levels at the end of each episode decreasing as a result of fewer episodes ending with a full storage buffer.

Training was performed on an Apple M2 Pro CPU. On this system, the environment trained at an average of 103k steps/core/hr, with a simulation-to-realtime ratio of 165.5 sim-days/core/hr. More complex environments can be expected to train at a slower — but still acceptable — rate.

5. Discussion

BSK-RL has been used in a variety of RL-based satellite tasking papers, demonstrating its flexibility and utility. The stability and configurability of the environments has allowed for rapid iteration on research questions, with less time spent on debugging and environment setup. A few recent papers and the utility of BSK-RL in their preparation are highlighted here:

In "Reinforcement Learning For Earth-Observing Satellite Autonomy With Event-Based Task Intervals" [44], BSK-RL is used to configure an AEOS tasking environment, demonstrating that RL can learn a policy that performs near-optimally with significantly less computation time than a MILP-based solver. BSK-RL enables the paper to perform ablation studies over different observation sizes and parameterizations with minimal effort. It also allows for easy deployment and analysis of the learned policy over various test cases by changing the scenario parameters. Even the MILP-based solutions [45] use BSK-RL due to its streamlined interface and high-fidelity simulation, providing a good test of the abstraction-to-real gap present in planning algorithms.

The paper "Using Enhanced Simulation Environments to Improve Reinforcement Learning for Long-Duration Satellite Autonomy" [17] performs a study on curriculum learning-like methods for training a satellite with resource constraints. The paper demonstrates that it may be beneficial to train an agent in a more challenging environment for greater fault tolerance. BSK-RL was used to be able to easily modify the environment difficulty by changing the satellite and environment parameters dictionaries.

"Intent Sharing for Emergent Collaboration in Autonomous Earth Observing Constellations" [21] explores the use of single-agent-trained policies in a multiagent scenario. For this study, the single-agent environment used for training is easily extended to a multiagent environment for evaluation through the addition of satellites. The communication module of BSK-RL is used to share data between agents, allowing for emergent collaboration between agents.

In "Learning For Satellite Autonomy Under Different Cloud Coverage Probability Observations" [46], an Earth-observing point target environment is extended to feature randomized cloud coverage. This environment with stochastic success for imaging tasks was implemented just by modifying the scenario and the rewarder, demonstrating the modularity of the BSK-RL package.

6. Conclusion

BSK-RL offers a high-fidelity, modular, and open-source environment for spacecraft tasking and control RL environments. Its architecture ensures that underlying physical dynamics and flight software are accurately represented within the RL environment, while an outer layer of abstractions allows for the definition of mission objectives and rewards; together complex missions can be defined around a physically realistic simulation. The high degree of configurability and the ease of training with off-the-shelf tools is demonstrated in an example environment; BSK-RL's use in a variety of other problem formulations further demonstrates its utility across various research questions. BSK-RL should be considered a valuable tool for advancing the application of RL to spacecraft tasking and control problems.

Acknowledgments

The BSK-RL package is an evolution of code originally developed by Andrew Harris and Adam Herrmann.

This work is supported by a NASA Space Technology Graduate Research Opportunity (NSTGRO) grant, 80NSSC23K1182. This work is also supported by the Air Force Research Lab grant FA9453-22-2-0050.

This work utilized the Alpine high-performance computing resource at the University of Colorado Boulder. Alpine is jointly funded by the University of Colorado Boulder, the University of Colorado Anschutz, Colorado State University, and the National Science Foundation (award 2201538).

Appendix: Additional Code Listings

Custom Simulator Models

An example of an extended dynamics model is shown below. This model composes two already-implemented dynamics models (`ContinuousImagingDynModel` for a nadir-pointing imaging instrument and `GroundStationDynModel` for ground station downlink) and adds custom properties to be included in the observation space.

```python
class ScanningDownlinkDynModel(
    ContinuousImagingDynModel,
    GroundStationDynModel
):
    @property
    def instrument_pointing_error(self) ->
    float:
        r_BN_P_unit = self.r_BN_P / np.linalg.
    norm(self.r_BN_P)
        c_hat_P = self.satellite.fsw.c_hat_P
        return np.arccos(np.dot(-r_BN_P_unit,
    c_hat_P))

    @property
    def solar_pointing_error(self) -> float:
        a = (
            self.world.gravFactory.spiceObject
    .planetStateOutMsgs[self.world.sun_index]
            .read()
            .PositionVector
        )
        a_hat_N = a / np.linalg.norm(a)
        nHat_B = self.satellite.sat_args["
    nHat_B"]
        NB = np.transpose(self.BN)
        nHat_N = NB @ nHat_B
        return np.arccos(np.dot(nHat_N,
    a_hat_N))
```

Overriding Satellite Parameters

An example of a dictionary of satellite parameter overrides and randomizers is shown below. The data storage capacity and fill rates are set and the power system parameters are set, among other parameters.

```python
sat_args=dict(
    # Data
    dataStorageCapacity=5000 * 8e6,  # bits
    storageInit=lambda: np.random.uniform(0.0,
     0.8) * 5000 * 8e6,
    instrumentBaudRate=0.5 * 8e6,
    transmitterBaudRate=-50 * 8e6,
    # Power
    batteryStorageCapacity=200 * 3600,  # W*s
    storedCharge_Init=lambda: np.random.
    uniform(0.3, 1.0) * 200 * 3600,
    basePowerDraw=-10.0,  # W
    instrumentPowerDraw=-30.0,  # W
    transmitterPowerDraw=-25.0,  # W
    thrusterPowerDraw=-80.0,  # W
    panelArea=0.25,
    # Attitude
    imageAttErrorRequirement=0.1,
    imageRateErrorRequirement=0.1,
    disturbance_vector=lambda: np.random.
    normal(scale=0.0001, size=3),  # N*m
    maxWheelSpeed=6000.0,  # RPM
    wheelSpeeds=lambda: np.random.uniform
    (-3000, 3000, 3),
    desatAttitude="nadir",
)
```

Environment Instantiation

Code for the complete instantiation of the RL environment used in the examples is given below, when combined with the code listings throughout the paper.

```python
env = SatelliteTasking(
    satellite=ScanningSatellite("Scanner-1",
    sat_args=sat_args),
    scenario=UniformNadirScanning(
    value_per_second=1 / (5 * 5700.0)),
    rewarder=ScanningTimeReward(),
    time_limit=5 * 5700.0,  # About 5 orbits
    failure_penalty=-1.0,
    terminate_on_time_limit=True,
)
```

References

[1] Seablom, M., Moigne, J. L., Kumar, S., Forman, B., and Grogan, P., "Real-Time Applications of the Nasa Earth Science "New Observing Strategy"," *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, Kuala Lumpur, Malaysia, 2022, pp. 5317–5320. https://doi.org/10.1109/IGARSS46834.2022.9883850.

[2] Wang, X., Wu, G., Xing, L., and Pedrycz, W., "Agile Earth Observation Satellite Scheduling Over 20 Years: Formulations, Methods, and Future Directions," *IEEE Systems Journal*, Vol. 15, No. 3, 2021, pp. 3881–3892. https://doi.org/10.1109/JSYST.2020.2997050.

[3] Lemaitre, M., Verfaillie, G., Jouhaud, F., Lachiver, J.-M., and Bataille, N., "Selecting and Scheduling Observations of Agile Satellites," *Aerospace Science and Technology*, Vol. 6, No. 5, 2002, pp. 367–381. https://doi.org/10.1016/S1270-9638(02)01173-2.

[4] Nag, S., Li, A. S., and Merrick, J. H., "Scheduling Algorithms for Rapid Imaging Using Agile Cubesat Constellations," *Advances in Space Research*, Vol. 61, No. 3, 2018, pp. 891–913. https://doi.org/10.1016/j.asr.2017.11.010.

[5] Cho, D.-H., Kim, J.-H., Choi, H.-L., and Ahn, J., "Optimization-Based Scheduling Method for Agile Earth-Observing Satellite Constellation," *Journal of Aerospace Information Systems*, Vol. 15, No. 11, 2018, pp. 611–626. https://doi.org/10.2514/1.I010620.

[6] Kim, J., Ahn, J., Choi, H.-L., and Cho, D.-H., "Task Scheduling of Agile Satellites with Transition Time and Stereoscopic Imaging Constraints," *Journal of Aerospace Information Systems*, Vol. 17, No. 6, 2020, pp. 285–293. https://doi.org/10.2514/1.I010775.

[7] Augenstein, S., "Optimal Scheduling of Earth-Imaging Satellites with Human Collaboration via Directed Acyclic Graphs," *The Intersection of Robust Intelligence and Trust in Autonomous Systems: Papers from the AAAI Spring Symposium*, 2014, pp. 11–16.

[8] Valicka, C. G., Garcia, D., Staid, A., Watson, J.-P., Hackebeil, G., Rathinam, S., and Ntaimo, L., "Mixed-Integer Programming Models for Optimal Constellation Scheduling given Cloud Cover Uncertainty," *European Journal of Operational Research*, Vol. 275, No. 2, 2019, pp. 431–445. https://doi.org/10.1016/j.ejor.2018.11.043.

[9] Wang, X., Gu, Y., Wu, G., and Woodward, J. R., "Robust Scheduling for Multiple Agile Earth Observation Satellites under Cloud Coverage Uncertainty," *Computers & Industrial Engineering*, Vol. 156, 2021, p. 107292. https://doi.org/10.1016/j.cie.2021.107292.

[10] Picard, G., Caron, C., Farges, J.-L., Guerra, J., Pralet, C., and Roussel, S., "Autonomous Agents and Multiagent Systems Challenges in Earth Observation Satellite Constellations," *International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021)*, 2021, pp. 39–44. https://doi.org/10.5555/3463952.3463961.

[11] Harris, A., Valade, T., Teil, T., and Schaub, H., "Generation of Spacecraft Operations Procedures Using Deep Reinforcement Learning," *Journal of Spacecraft and Rockets*, Vol. 59, No. 2, 2022, pp. 611–626. https://doi.org/10.2514/1.A35169.

[12] Eddy, D., and Kochenderfer, M., "Markov Decision Processes For Multi-Objective Satellite Task Planning," *2020 IEEE Aerospace Conference*, IEEE, Big Sky, MT, USA, 2020, pp. 1–12. https://doi.org/10.1109/AERO47225.2020.9172258.

[13] Hadj-Salah, A., Verdier, R., Caron, C., Picard, M., and Capelle, M., "Schedule Earth Observation Satellites with Deep Reinforcement Learning," , Nov. 2019.

[14] Zhao, X., Wang, Z., and Zheng, G., "Two-Phase Neural Combinatorial Optimization with Reinforcement Learning for Agile Satellite Scheduling," *Journal of Aerospace Information Systems*, Vol. 17, No. 7, 2020, pp. 346–357. https://doi.org/10.2514/1.I010754.

[15] Chun, J., Yang, W., Liu, X., Wu, G., He, L., and Xing, L., "Deep Reinforcement Learning for the Agile Earth Observation Satellite Scheduling Problem," *Mathematics*, Vol. 11, No. 19, 2023, p. 4059. https://doi.org/10.3390/math11194059.

[16] Herrmann, A., and Schaub, H., "Reinforcement Learning for the Agile Earth-Observing Satellite Scheduling Problem," *IEEE Transactions on Aerospace and Electronic Systems*, 2023, pp. 1–13. https://doi.org/10.1109/TAES.2023.3251307.

[17] Stephenson, M., Mantovani, L., Phillips, S., and Schaub, H., "Using Enhanced Simulation Environments to Improve Reinforcement Learning for Long-Duration Satellite Autonomy," *AIAA Science and Technology Forum and Exposition (SciTech)*, Orlando, Florida, 2024-01-08/2024-01-12. https://doi.org/10.2514/6.2024-0990.

[18] Li, P., Wang, H., Zhang, Y., and Pan, R., "Mission Planning for Distributed Multiple Agile Earth Observing Satellites by Attention-Based Deep Reinforcement Learning Method," *Advances in Space Research*, 2024, p. S0273117724005465. https://doi.org/10.1016/j.asr.2024.06.003.

[19] Wang, H., Yang, Z., Zhou, W., and Li, D., "Online Scheduling of Image Satellites Based on Neural Networks and Deep Reinforcement Learning," *Chinese Journal of Aeronautics*, Vol. 32, No. 4, 2019, pp. 1011–1019. https://doi.org/10.1016/j.cja.2018.12.018.

[20] Herrmann, A., Stephenson, M. A., and Schaub, H., "Single-Agent Reinforcement Learning for Scalable Earth-Observing Satellite Constellation Operations," *Journal of Spacecraft and Rockets*, 2023, pp. 1–19. https://doi.org/10.2514/1.A35736.

[21] Stephenson, M., Mantovani, L., and Schaub, H., "Intent Sharing for Emergent Collaboration in Autonomous Earth Observing Constellations," *AAS GN&C Conference*, Breckenridge, CO, Februrary 2, 2024.

[22] Piccinin, M., Lunghi, P., and Lavagna, M., "Deep Reinforcement Learning-based Policy for Autonomous Imaging Planning of Small Celestial Bodies Mapping," *Aerospace Science and Technology*, Vol. 120, 2022, p. 107224. https://doi.org/10.1016/j.ast.2021.107224.

[23] Chen, Z., Cui, H., and Tian, Y., "Autonomous Maneuver Planning for Small-Body Reconnaissance via Reinforcement Learning," *Journal of Guidance, Control, and Dynamics*, 2024, pp. 1–13. https://doi.org/10.2514/1.G008011.

[24] Oestreich, C. E., Linares, R., and Gondhalekar, R., "Autonomous Six-Degree-of-Freedom Spacecraft Docking with Rotating Targets via Reinforcement Learning," *Journal of Aerospace Information Systems*, Vol. 18, No. 7, 2021, pp. 417–428. https://doi.org/10.2514/1.I010914.

[25] Sharma, K. P., Kumar, I., Singh, P. P., Anbazhagan, K., Albarakati, H. M., Bhatt, M. W., Ziyadullayevich, A. A., Rana, A., and A, S. S., "Advancing Spacecraft Rendezvous and Docking through Safety Reinforcement Learning and Ubiquitous Learning Principles," *Computers in Human Behavior*, Vol. 153, 2024, p. 108110. https://doi.org/10.1016/j.chb.2023.108110.

[26] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D., "Deep Reinforcement Learning That Matters," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32, No. 1, 2018. https://doi.org/10.1609/aaai.v32i1.11694.

[27] Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., de Cola, G., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, A. J. S., and Younis, O. G., "Gymnasium," , Oct. 2023.

[28] Terry, J. K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C., Williams, N. L., Lokesh, Y., and Ravi, P., "PettingZoo: Gym for Multi-Agent Reinforcement Learning," , Oct. 2021.

[29] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M., "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research*, Vol. 47, 2013, pp. 253–279. https://doi.org/10.1613/jair.3912.

[30] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M., "Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents," , Nov. 2017. https://doi.org/10.48550/arXiv.1709.06009.

[31] Todorov, E., Erez, T., and Tassa, Y., "MuJoCo: A Physics Engine for Model-Based Control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Vilamoura-Algarve, Portugal, 2012, pp. 5026–5033. https://doi.org/10.1109/IROS.2012.6386109.

[32] Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T., "Mastering Diverse Domains through World Models," , Jan. 2023.

[33] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D., "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning," *Nature*, Vol. 575, No. 7782, 2019, pp. 350–354. https://doi.org/10.1038/s41586-019-1724-z.

[34] Allen, R., Rachlin, Y., Ruprecht, J., Loughran, S., Varey, J., and Viggh, H., "SpaceGym: Discrete and Differential Games in Non-Cooperative Space Operations," *2023 IEEE Aerospace Conference*, Zenodo, 2023, pp. 1–12. https://doi.org/10.5281/ZENODO.11256264.

[35] Kenneally, P. W., Piggott, S., and Schaub, H., "Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework," *Journal of Aerospace Information Systems*, Vol. 17, No. 9, 2020, pp. 496–507. https://doi.org/10.2514/1.I010762.

[36] Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I., "RLlib: Abstractions for Distributed Reinforcement Learning," , Jun. 2018.

[37] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N., "Stable-Baselines3: Reliable

Reinforcement Learning Implementations," *Journal of Machine Learning Research*, Vol. 22, 2021, pp. 1–8.

[38] Bradtke, S., and Duff, M., "Reinforcement Learning Methods for Continuous-Time Markov Decision Problems," *Advances in Neural Information Processing Systems*, Vol. 7, MIT Press, 1994.

[39] Menda, K., Chen, Y.-C., Grana, J., Bono, J. W., Tracey, B. D., Kochenderfer, M. J., and Wolpert, D., "Deep Reinforcement Learning for Event-Driven Multi-Agent Decision Processes," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 20, No. 4, 2019, pp. 1259–1268. https://doi.org/10.1109/TITS.2018.2848264.

[40] Brandonisio, A., Capra, L., and Lavagna, M., "Deep Reinforcement Learning Spacecraft Guidance with State Uncertainty for Autonomous Shape Reconstruction of Uncooperative Target," *Advances in Space Research*, 2023, p. S0273117723005276. https://doi.org/10.1016/j.asr.2023.07.007.

[41] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," , Aug. 2017.

[42] Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U., "Safe Reinforcement Learning via Shielding," , Sep. 2017.

[43] Nazmy, I., Harris, A., Lahijanian, M., and Schaub, H., "Shielded Deep Reinforcement Learning for Multi-Sensor Spacecraft Imaging," *2022 American Control Conference (ACC)*, IEEE, Atlanta, GA, USA, 2022, pp. 1808–1813. https://doi.org/10.23919/ACC53348.2022.9867762.

[44] Stephenson, M., and Schaub, H., "Reinforcement Learning For Earth-Observing Satellite Autonomy With Event-Based Task Intervals," *AAS Rocky Mountain GN&C Conference*, Breckenridge, CO, 2024.

[45] Stephenson, M., and Schaub, H., "Optimal Target Sequencing In The Agile Earth-Observing Satellite Scheduling Problem Using Learned Dynamics," *AAS/AIAA Astrodynamics Specialist Conference*, Big Sky, MT, 2023.

[46] Mantovani, L. Q., Nagano, Y., and Schaub, H., "Reinforcement Learning For Satellite Autonomy Under Different Cloud Coverage Probability Observations," *AAS Astrodynamics Specialist Conference*, Broomfield, CO, Aug. 11-15,2024.