AAS 20–117

DEEP ON-BOARD SCHEDULING FOR AUTONOMOUS ATTITUDE GUIDANCE OPERATIONS

Andrew Harris[*] and Hanspeter Schaub[†]

Increasingly complex space missions have motivated the development of autonomous mission guidance approaches capable of dealing with high-dimensional, continuous observation and action spaces. Deep reinforcement learning (DRL) techniques are a rising area of research for dealing with such problems, but at present lack clear methods for verification or validation, especially in the context of spacecraft operations. This work identifies a specific problem architecture for addressing a high-level attitude mode guidance problem on-board through the use of a pre-trained learning agent using contemporary strategies for safety and verification from the deep learning community. Additionally, high-performance, open-source space-specific simulation tools derived from the AVS Basilisk astrodynamics simulation package are presented and discussed. The resulting end-to-end development and verification pipeline is presented against other approaches and compared on the basis of accuracy, computational efficiency, and safety.

INTRODUCTION

Operating spacecraft without human operators in the loop has become a major enabler for future mission architectures ranging from deep-space asteroid sample return to large-scale Earth-orbiting constellations.[11] While decades of development have yielded notable successes in the development of decision support software for operators[8] or on-board observation planning,[5–7] these approaches require substantial development efforts and may struggle to scale as the number of parameters considered increases. At the same time, the machine learning community has renewed its focus Reinforcement Learning techniques that leverage the capabilities of deep neural networks (termed "deep" reinforcement learning or DRL) to address similar high-dimensional decision problems, such as those presented by strategy games[20] or multi-agent coordination problems.[15] This work explores the application of DRL approaches to a representative attitude mode guidance problem.

At present, examples of spacecraft autonomy typically fall into two categories: rule-based autonomy and optimization-based autonomy. Rule-based autonomy treats a space-

[*]Research Assistant, Ann and H.J. Smead Department of Aerospace Engineering Sciences, University of Colorado Boulder, Boulder, CO, 80309 USA. AAS Member, AIAA Member.

[†]Glenn L. Murphy Chair of Engineering, Smead Department of Aerospace Engineering Sciences, University of Colorado, 431 UCB, Colorado Center for Astrodynamics Research, Boulder, CO 80309-0431. AAS Fellow, AIAA Fellow.

craft as a state machine consisting of a set of mode behaviors and defined transitions between modes. Pioneered by missions like Deep Impact,[16] and currently used by missions such as the PlanetLabs constellation,[10] spacecraft using rule-based autonomy transition between operational and health-keeping modes (charging, momentum-exchange device desaturation) autonomously without ground contact. In contrast, optimization-based autonomy treats the spacecraft and its mission in the framework of constrained optimization, with the spacecraft's hardware and trajectory acting as constraints and metrics of mission return–images taken, communication link uptime, or other criteria–are the values being optimized. In contrast to rule-based autonomy, optimization-based autonomy typically requires large amounts of computing power throughout the mission life-cycle. Examples of this work include the Applied Physics Laboratory's SciBox software library (used to generate MESSENGER mode sequences) and the ASPEN mission planning suite developed by the Jet Propulsion Laboratory and applied to the Earth Observing-1 mission.[7]

Owing to their successes in solving other high-dimensional, complex online decision-making problems, Deep RL strategies appear well-suited to the spacecraft operations management problem; the scaling properties of deep networks allows them to tackle high-dimensional, non-convex problems, while trained neural networks themselves are relatively quick to execute in comparison to optimization strategies. A small collection of other works in the application of machine learning techniques to spacecraft problems exists in the recent literature, mostly focusing on the application of learning approaches to control problems in uncertain environments. Several works, such as References[13] and,[9] consider reinforcement learning in the context of autonomous aerobraking planners, with mixed results. Others explore machine learning techniques for asteroid proximity operations[12] or autonomous lunar landing.[17] This work builds on prior work in high-level spacecraft tasking and planning,[14] creating a problem in the domain of attitude mode guidance that considers high-level mission objectives, traditional guidance considerations (such as mis-modeled dynamics), and spacecraft health constraints.

While few guarantees outside of bounds on performance improvement are available for deep reinforcement learning strategies, new approaches that augment pure DRL strategies to provide safety and correctness guarantees have appeared in the literature. Shielded learning[2] uses a low-dimensional representation of safety constraints as a basis for forming a "shield" algorithm that prevents a trained (or in-training) agent from taking unsafe actions. As a result, shielded learning approaches combine the scalability of traditional DRL with the safety gauruntees provided by correct-by-construction control approaches. At the same time, statistical verification theory has also made strides in providing autonomous, computationally expedient tools for determining the correctness and robustness of black-box controllers while minimizing the use of computationally expensive Monte Carlo analysis. The work presented herein will examine both of these strategies in the context of a spacecraft operations deep reinforcement learning pipeline, and compare their performance against a naive implementation of deep learning for the spacecraft tasking problem. This work will include three following components. First, the high-level attitude mode guidance problem is specified considering relevant problem dynamics and spacecraft-side limitations. Next, a deep learning agent incorporating both shielded learning and statistical verification is presented and analyzed for performance and computational efficiency on a baseline problem, demonstrating a proof-of-concept for using learning agents to conduct high-level control. Special consideration is given to the selection of hyperparameters, such as network size and activation function, for this baseline analysis. Finally, the learning agent is tested against parametric variations in disturbance dynamics to validate the approach.

PROBLEM STATEMENT

Command and Control Framework

Traditional spacecraft operations planning and execution is a complex, multi-step process with many stakeholders which relies heavily on expert knowledge. For reference, a generic version of this paradigm is presented here. First, mission stakeholders specify mission objectives and a reference mission trajectory. Given this trajectory and a set of desired tasks, a set of activities or operational modes are defined and scheduled as spacecraft resources (power, fuel, compute time) and mission resources (observation, maneuver, or communication windows) permit. Finally, these plans are converted into an action sequence, up-linked to a spacecraft, and executed by on-board software. Simultaneously, teams of human operators typically monitor mission execution and spacecraft health parameters and intervene when parameters fall outside of a defined specification, either directly by changing the current action sequence or indirectly by initiating a re-planning sequence. While this process or processes like it have been used successfully for decades, it relies heavily on human expertise to create priorities, construct action sequences, and verify spacecraft behavior. In the search for future autonomy approaches, it is desirable

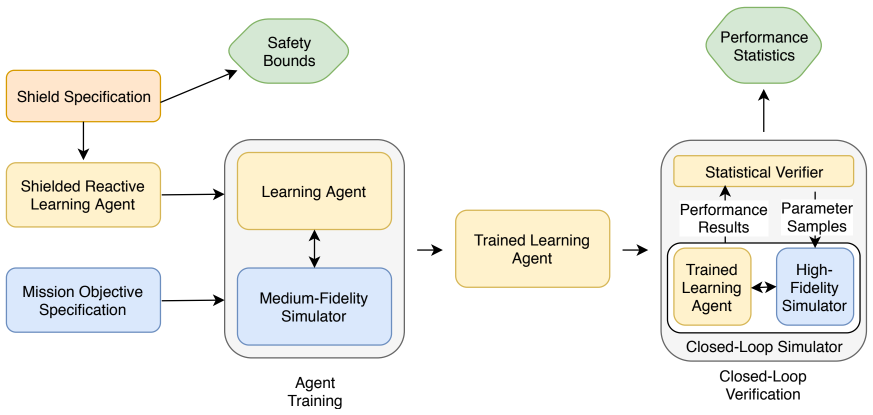


Figure 1: End-to-End training pipeline for autonomous operations agents.

to both replicate existing capabilities in the realm of rule-based and optimization-oriented autonomy while improving their extensibility, robustness to un-modeled dynamics, and computational burden. To provide a feasible scope, this work specifically considers the mission-level decision-making problem wherein sub-plans (“modes”) have already been identified, either by some other planning routine or by designers pre-flight. In this context, a decision-making agent must account not only for mission objectives, but also the constraints imposed by spacecraft hardware, orbital and attitude mechanics, and uncertainty regarding known or unknown environmental parameters.

A common framework for representing and addressing such problems are Partially-Observable Markov Decision Processes, which compactly represent the problems facing a software agent acting in an evolving environment according to some higher-level objective.[3] The mathematics of such processes, and challenges associated with them, are reviewed briefly here.

A model of several time-steps of a classical POMDP is presented in Fig. 2, and discussed further here. As in traditional Markov Decision Processes (MDPs), the state in a POMDP is updated by a transition function F, and at any given time can be computed as a function of the previous state and the most recent action taken by the considered agent(s):

$$s_k = F(s_{k-1}, a_{k-1}) \tag{1}$$

This state s_k is observed by the agent according to some observation function H:

$$o_k = H(s_k) \tag{2}$$

Given an observation o_k of the state, the agent then selects an action a_k to influence the future state according to some policy π:

$$a_k = \pi(o_k) \tag{3}$$

While these transition functions represent physical or software-defined process dynamics, the objective of an agent is ultimately motivated by a reward function R:

$$r_k = R(s_{k-1}, a_{k-1}, s_k) \tag{4}$$

Taken together, these components form a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, F, H, \mathcal{R}.)$

The objective of a software agent within a POMDP is to select a policy π that maximizes its realized reward. While the general POMDP case places no restrictions on the nature of any of the transition functions or states, the consideration of infinite-dimensional, continuous state and action spaces can be extremely computationally intensive. For this reason, many applied autonomy approaches that leverage POMDPs perform some degree of discretization to their state or action space. Additionally, it is noted that POMDPs attempt to describe holistic, system-level problems within a unified framework that is theoretically related to but practically divorced from traditional estimation and control approaches. For these reasons, POMDP-based approaches to autonomy are most frequently studied in cases where traditional estimation and controls approaches are not readily tractable, including human-assisted machine decision-making[15] or multi-vehicle coordination problems.[18]

A representative space orbit operations scenario is presented and defined as a partially-observable Markov decision process (POMDP) to be solved using Proximal Policy Optimization (PPO) For a spacecraft, the general high-level autonomy POMDP can be stated as follows. Given the constraints of orbital dynamics, on-board hardware, and pre-defined software behaviors, select the sequence of behaviors that best satisfies mission objectives. This approach is described in greater detail in.[14]

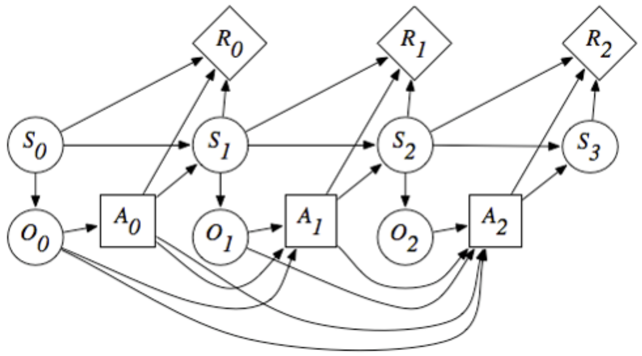


Figure 2: Sequential Partially Observable Markov Decision process framework for representing decision problems.

Deep Reinforcement Learning

Astrodynamics and spacecraft-planning problems are typically considered in the context of continuous estimation and control, as many of the processes facing such systems are infinite-dimensional with well-understood, reasonably accurate models. Unfortunately, the high-level relationships between spacecraft actions and the satisfaction of mission objectives is less analytically tractable, and frequently mixes discrete reward states (such as whether a geological feature has been imaged) with continuous ones (such as the management of spacecraft power states). Reinforcement Learning (RL) techniques represent one class of algorithms for addressing decision processes which lack analytically tractable models; however, traditional RL techniques require discrete state and action spaces. Prior work[13] has shown that accurate discrete state spaces for spacecraft decision problems can be extremely large and therefore infeasible to explore. To address these shortcomings, Deep Reinforcement Learning (DRL) techniques use deep neural networks as function approximators in place of tables and can therefore learn on continuous state or action spaces without discretization.

Reinforcement Learning techniques are intended to solve general Markov Decision Processes (MDPs), which are simplified forms of POMDPs without the issue of observation functions. The goal of reinforcement learning is to find an optimal policy π^* that maximizes the expected future reward of the agent. The optimal policy, π^*, is the policy with the largest expected sum or rewards or value function. The cumulative value, V, of a given state is provided by the discounted sum of the rewards from the current infinitely into the future and is given below:

$$V(s_0, s_1, s_2, ...) = \sum_{t=0}^{\infty} \gamma^t r_t \qquad 0 \leq \gamma < 1 \tag{5}$$

where γ is the reward discount factor. This term weights the importance of future rewards relative to the current reward. Given this framework, the optimal policy is that which maximizes the expected discounted future reward.

$$\pi^* = \arg\max_{\pi} \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{6}$$

This leads to another expression for the cumulative value function referred to as Bellman's Equation:

$$V(s) = R(s) + \gamma \max_{a} \sum_{s'} p(s'|s,a)V(s') \tag{7}$$

where $p(s'|s,a)$ is the probability of the agent being in state, s', after performing action, a, in state s.

This work uses Proximal Policy Optimization[19] as implemented by the `stable-baselines` Python package [*], an extended variant of Trust-Region Policy Optimization. PPO uses a loss function that penalizes the learning agent from dramatically changing its network weights from iteration to iteration using advantage estimation and a clipping function:

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta),\ 1-\epsilon,\ 1+\epsilon)\hat{A}_t)] \tag{8}$$

where θ represents the policy parameters, $\hat{E}_t$ represents the learned expectation over timesteps, r_t is the ratio of the probability under the new and old policies, $\hat{A}_t$ is the estimated advantage at t, and ϵ is a hyperparameter representing the clipping range. The advantage function is defined as the difference between the state-action value function $Q^{\pi}(s,a)$ and the state value function $V^{\pi}(s)$. This approach has been shown to produce faster, more reliable convergence than other results, and represents the state-of-the-art in model-free deep reinforcement learning.

Agent Implementation Frameworks

A major assumption in our formulation of the spacecraft control problem as a (PO)MDP shown in Eqn. 11 is the discretization of time, which–when combined with the mechanics of learning as described in Section –results in decision-making agents that can only *react* to current observations, as shown in Fig. 3. Rather than utilizing a specific plan or strategy, all relevant planning and strategy information is encoded in the deep network utilized by the agent. In practice, evaluating neural networks is nearly constant-time and can be readily hardware-accelerated, making this implementation attractive for future on-board use where system information is readily available and humans are already out of the loop.

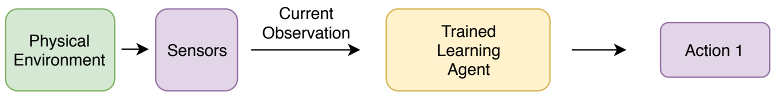


Figure 3: Sequential decision-making agent architecture.

At the same time, many existing systems assume that discrete sets of actions will be periodically up-linked from the ground and lack the on-board processing power to evaluate

[*]https://github.com/hill-a/stable-baselines

a neural network. For these systems, an architecture which uses a ground-side simulator to propagate forward existing observations and actions is proposed as shown in Fig. 4. The incorporation of a simulator allows for the agent to make “future” decisions based on current knowledge and plan ahead. This architecture is also attractive for near-term implementation, as it allows human operators to verify and validate action sequences in advance of execution.

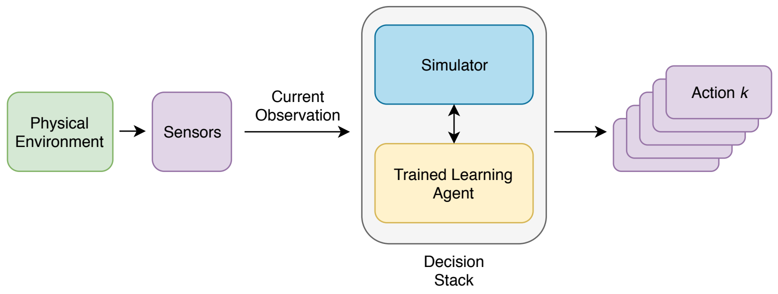


Figure 4: Planning architecture using a sequential decision-making agent.

Examination of the properties and benefits of planning versus reactive agents is left outside the scope of this work, which focuses on establishing training and safety properties for DRL-based sequential decision-making agents for spacecraft command and control.

Safety Guarantees

Safety in the face of uncertain spacecraft performance, environmental parameters, and operating sequences is a critical requirement for future autonomy architectures. While some reinforcement learning techniques can bound their performance with respect to a reward function within an MDP, there are virtually none which can guarantee safety on their own. In practice, this is dealt with through reward engineering; unsafe action or state combinations are given large costs or penalties to achieved reward. This approach has several key disadvantages: many problems for which reinforcement learning is well-suited have complex environment/reward interactions, which makes manual reward engineering difficult; when reward engineering is feasible, it does not prevent the agent from taking unsafe actions in conditions outside the training set presented by its environment; finally, there is no quantifiable boundary or degree of safety provided through reward engineering. These shortcomings have motivated the search for alternative approaches to safety that can be combined with common DRL approaches.

Reactive synthesis is one category of techniques that can provide performance bounds and guarantees for controllers on specified systems. In general, reactive synthesis algorithms operate on discrete, known, finite systems and attempt to produce behavior on such systems that satisfies a specification written in a temporal logic language, such as Linear Temporal Logic (LTL). Also described as “correct-by-construction” approaches, reactive synthesis algorithms only produce control policies that meet a given specification; if the specification cannot be met on the current system, no policy will be produced, allowing for designers to check feasibility before implementation. While powerful for addressing systems with discrete, finite, known dynamics, reactive synthesis approaches scale poorly with system and specification complexity, which limits their applicability for solving general spacecraft planning problems, which are difficult to discretize to sufficient fidelity.[13]

Shielded learning techniques[2] combine common DRL approaches with reactive synthesis-based shields to combine the power of black-box optimization with formal guarantees of safety. Shielded learning depends on the construction of a coarse, finite-state safety MDP from the original MDP the learning agent is intended to solve that is conservative with respect to the original environment's dynamics and the safety specification, yet limited enough that reactive synthesis can be applied to it. Next, a safety specification is created using Linear Temporal Logic which encapsulates all desired safety conditions and provided as an input to a reactive synthesis algorithm, such as a two-player game, which produces a discrete, state-dependent strategy. Finally, this strategy is implemented alongside the learning agent as shown in Figure 5; in this implementation, the shield accepts observations of the current system state and the action attempted by the learning agent, and permits the action only if it aligns with the shield's strategy. This implementation architecture is applicable to both training and on-line use of the sequential decision agent, allowing it to provide safety boundaries during mission execution.

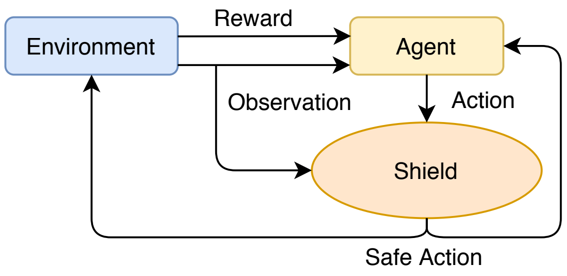


Figure 5: Post-Posed shielded reinforcement learning framework.

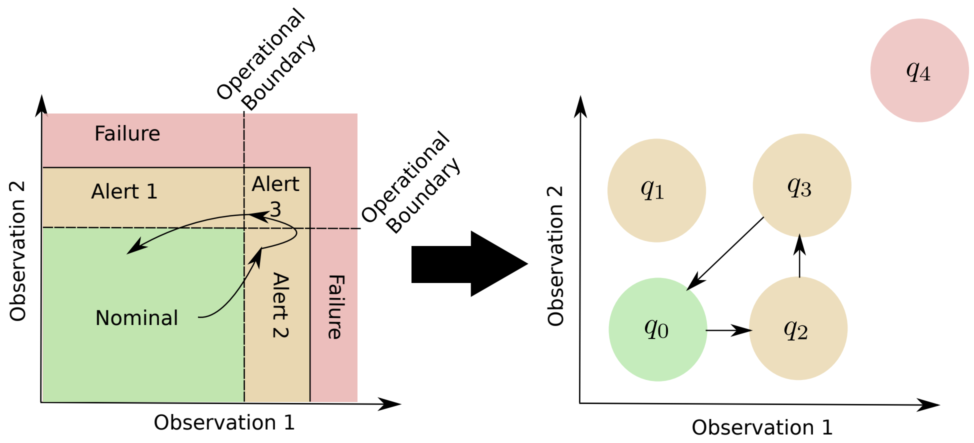


Figure 6: Conversion from continuous states to a discrete safety MDP.

An example of this transformation in practice is shown for a system with two safety-critical dimensions in Fig. 6. Mission designers first identify state combinations that represent mission failure, such as depleting the spacecraft's battery or allowing reaction wheels to spin up beyond manufacturer's specifications. In addition to the hard safety boundaries, operators and mission planners typically incorporate additional boundaries to act as margins of safety against actual failure; these are represented by the dashed lines labeled "operational boundary," which are used to define warning states. While in this boundary, operators would be expected to take immediate action to return the system to safe, nominal operating conditions. In this view, the system's behavior can be plotted on a phase-plot, where individual samples of the system's true trajectory are represented as curves in the observation variable space. The continuous, bounded system creates a natural framework for the construction of a safety MDP, wherein each warning state becomes a discrete state, including products of warning states. It is important that the safety MDP contain all information necessary for the system to operate safely, which may require the inclusion of states which are not themselves safety risks but which affect the performance of actions necessary for the safety of the system. This process results in a discrete safety MDP which exists in parallel with the continuous POMDP.

Verification Strategy

Training AI agents necessarily involves the use of models that may contain parametric uncertainties. A key concern with deep reinforcement learning approaches is in preventing over-fitting to certain model behaviors and parameters. To verify that a given AI operations agent is functional within its specification under plausible conditions, a robust verification strategy is required. Here, a two-part verification strategy is used: first, the agent is employed in a higher-fidelity model of the system to check for signs of over-fitting; next, a statistically informed robustness metric is used to allocate these high-fidelity simulation runs over the set of plausible parametric uncertainties.

The field of statistical verification theory is especially applicable to complex cyber-physical systems for which strict analytical guarantees on system performance are not available. Unlike classical model-checking techniques, which both depend on deterministic system models and provide only binary success or failure metrics, statistical verification techniques utilize samples of system performance generated by a simulator alongside a specification to automatically test system performance.

Simulation Framework

Both the training process for DRL-based operations agents and the verification framework require the ability to simulate a space mission to high fidelity. DRL techniques in particular can struggle when transferring from simulated to real experiences due to the simulation gap, as DRL agents can over-fit on specific attributes of low-fidelity simulators which do not generalize to the real world. In the same vein, verification techniques require the existence of high-fidelity, trusted simulation capability which adequately captures the behavior of the real system. For spacecraft, this requires the ability to simulate not only traditional astrodynamics components (orbital and attitude dynamics), but also the behavior of flight software components.

The Basilisk astrodynamics simulation package represents an ideal toolset for both of these applications. Specifically, Basilisk provides:

1. **High Fidelity Astrodynamics:** The Basilisk dynamics engine can simulate fully-coupled multi-body dynamics in tandem with GPU-accelerated orbital dynamics,[1] allowing for the simulation of second- and third-order effects like attitude/orbit coupling, fuel slosh, and flexing panels.

2. **Flight Software Simulation/Integration:** Developed as a tool to aid flight software development by providing a flight-like environment for testing, Basilisk provides first-class support for the integration of flight software components.

3. **Computational Performance:** Compute-heavy code is written in C/C++ and is highly performant as a result; even with tasks like image generation in the loop, BSK-based simulations are thousands of times faster than real-time, allowing for rapid generation of samples for both DRL and verification algorithms.

4. **Integration with common ML/RL frameworks:** Basilisk is written with SWIG and provides a Python API for setting up, executing, and analyzing simulations, which allows it to be integrated with other common ML/RL packages (Tensorflow, Keras, gym, `scikit-learn`).

To facilitate the integration of Basilisk with other machine learning tools, a library of OpenAI `gym` environments which utilize Basilisk for spacecraft simulation has been created and opened to the public. This library supports common DRL frameworks with Python APIs, such as OpenAI's

PERFORMANCE COMPARISON

To demonstrate the viability and applicability of deep and shielded learning techniques, this section applies them to a reference problem implemented using the Basilisk deep learning framework and compares their performance in both training (time to convergence, performance with respect to the reward function) and execution (qualitative evaluation of safety).

Reference Mission Operations Problem

For the purposes of this work, a scenario consisting of a single spacecraft conducting ground observations of Earth is considered. In general, the goal of the operations agent is to maximize both the time spent pointing at the ground and the accuracy of that ground-pointing mode; as such, a reward function which diminishes smoothly as the spacecraft attitude varies away from the ground-pointing reference is selected as

$$R_s = \frac{1}{1 + |\boldsymbol{\sigma}_{\text{err}}|} \text{ if } a = \text{Science} \tag{9}$$

Pointing is accomplished through the use of three reaction wheels with randomized initial biases; attitude determination is accomplished using a truth-plus-noise simulation of an ideal attitude estimator. In addition to managing the science pointing mode, the spacecraft operations agent must also ensure that the system remains power-positive by pointing the spacecraft's body-fixed solar panel towards the sun. The spacecraft's power consumption is modeled using a simple net power process:

$$\dot{J} = W_{\text{in}} - W_{\text{out}} \tag{10}$$

where J is the total energy stored by the spacecraft's battery, W_{in} is the power produced by the solar panel which is assumed to follow a cosine law, and W_{out} is the constant load power drawn by the spacecraft; for the purposes of this work, the load power is assumed to be constant.

To further complicate the operations problem, reaction wheel saturation is also modeled. In LEO, a primary source of disturbance torques for spacecraft occur from interactions with planetary atmospheres. To this end, the spacecraft geometry is considered as a standard box-and-wing model with a large offset area representing the solar panel. Left uncorrected, reaction wheel speeds would increase to counteract the aerodynamic torques until they saturate, rendering the spacecraft uncontrollable. To desaturate the wheels, a set of RCS thrusters and a wheel desaturation algorithm are implemented as a third and final flight mode. This mode sets the attitude reference towards the sun, but periodically pulses the thrusters to reduce the wheel momentum. Importantly, this mode is constructed using a pre-existing desaturation algorithm that assumes a small body angular rate when computing thruster firing sequences; when entered before the attitude control system can stabilize the system, this mode produces destabilizing behavior. This type of constraint is representative of one the real-world challenges of incorporating strategies for autonomy around existing flight software stacks and operations procedures.

$$P = \begin{cases} s & = \{\boldsymbol{r} \in \mathbb{R}^3,\ \dot{\boldsymbol{r}} \in \mathbb{R}^3,\ \boldsymbol{\sigma}_{BN} \in \mathbb{O}^3, \boldsymbol{\omega}_{BN} \in \mathbb{R}^3, \boldsymbol{\omega}_{RW} \in \mathbb{R}^3, \text{J} \in \mathbb{R}^1\} \\ o & = \{\sigma_{BN} \in \mathbb{O}^3, \boldsymbol{\omega}_{BN} \in \mathbb{R}^3, \boldsymbol{\omega}_{RW} \in \mathbb{R}^3, \text{J} \in \mathbb{R}^1\} \\ a & = \{\text{Mission}, \text{Sun Pointing},\ \text{Desaturation}\} \\ T & = \{f_{\text{Mission}},\ \ f_{\text{Sun Pointing}},\ f_{\text{Desaturation}}\} \\ R & = \{R_s, -50 \text{ if } J = 0 \text{ or } |\boldsymbol{\omega}_R W| > 250\frac{rad}{s}\} \end{cases} \tag{11}$$

The abstract MDP described by Equation 11 represents a command and control problem for a single spacecraft in LEO with hardware constraints and is used as a reference problem. For training, the initial conditions are drawn from uniform random distributions over a range of LEO orbits; similarly, the spacecraft's internal states are randomized to ensure coverage over this space. A summary of the MDP's parameters is shown in Table 1.

Additionally, the parameters of the safety MDP are listed in Table 2.

Table 1: Initial conditions for the real-valued MDP; $\mathcal{U}$ represents a uniform distribution.

Variable	Value
r_{eq}	3396.19 km
a	$r_{eq} + 400.0km$
e	$\mathcal{U}(0, 0.5)$
i	$\mathcal{U}(-90°, 90°)$
ω	$\mathcal{U}(0°, 360°)$
Ω	$\mathcal{U}(0°, 360°)$
ν	$\mathcal{U}(0°, 360°)$
$\boldsymbol{\sigma}_{BN}$	$\mathcal{U}(\mathbf{0}, \mathbf{1})$
$\boldsymbol{\omega}_{BN}$	$\mathcal{U}(\mathbf{0}$ rad/s, $\mathbf{0}.1$ rad/s)
$\boldsymbol{\omega}_{BN}$	$\mathcal{U}(\mathbf{0}, \mathbf{0}.1)$
$\boldsymbol{\omega}_{RW}$	$\mathcal{U}(-600$ RPM, 600 RPM)
J_{stored}	$\mathcal{U}(5$ W-Hr, 10 W-Hr)
t_{mode}	3 minutes
T_{max}	540 modes

Table 2: Safety MDP labelling parameters

Observed Variable	Operational Limit	Safety Limit
$\lvert\omega_{BN}\rvert$	0.05 rad/s	N/A
$\lvert\omega_{RW}\rvert$	1,000 RPM	1,500 RPM
J_{stored}	5 W-Hr	0 W-Hr

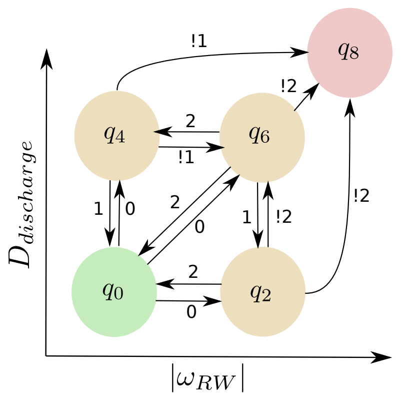


Figure 7: Safety MDP contructed for the LEO attitude mode planning simulator. $D_{discharge}$ represents the depth of discharge and is inversely analogous to J. Modes relating to "tumble" states with large body rates are omitted for clarity.

Shield Construction

To apply the shielded learning technique to space mission operations, a simplified version of the mission POMDP is first constructed using a-priori knowledge. Here, alert states are defined using the operational limits found in Table 2. These limits are applied to transform the continuous-time, continuous-state system described by Equation 11 into a simplified, discrete MDP in the observed variables, represented graphically in Fig. 7. This MDP is stated as P_{disc}:

$$P = \begin{cases} s & = \{\boldsymbol{\omega}_{BN} \in \{\text{nominal, high}\},\ |\omega_{RW}| \in \{\text{nominal, alert, failure}\}, \text{J}\in \{\text{nominal, low, failure}\} \\ o & = \{q \in \{q_0, q_1, ...q_7, q_8\} \\ a & = \{\text{Mission, Sun Pointing, Desaturation}\} \\ T & = \{f_{\text{Mission}},\ \ f_{\text{Sun Pointing}},\ f_{\text{Desaturation}}\} \\ R & = \{\emptyset\} \end{cases} \tag{12}$$

While substantially smaller than the continuous state POMDP, the safety MDP encodes important information; for example, desaturation events are only feasible when the spacecraft is not in a tumbling state, and tumbling states themselves do not lead to failure unless the battery charge or wheel speed are already near the failure criteria. In addition, the various state combinations that lead to failure are lumped into q_8 for brevity; this permits the use of the simple LTL specification

$$\varphi = G(\neg\text{“fail”}) \tag{13}$$

which is represented using the Büchi automaton shown in Fig. 8, and can be understood in English as "globally never allow the state to reach the failure state."

To solve this safety game, the game itself was implemented as a stochastic Markov game (`smg`) within the PRISM-games solver. In this case, PRISM-games solves the safety game using Value Iteration.[4] PRISM-games then saves the shield strategy as a `.adv` file, which encodes the state-action strategy which maximizes the probability of remaining safe. For

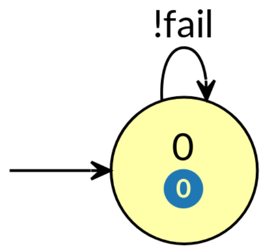


Figure 8: The one-state Büchi automaton representing the safety specification for the system.

this work, the resulting strategy is memoryless and state-based, making it especially amicable to on-line implementation.

To use this adversary strategy, the `stable-baselines` implementation of PPO2 was extended to conform to the post-posed shield framework shown in Fig. 5.

Training Results

To provide a comparison between the shielded and unshielded approaches to DRL-based spacecraft autonomy, three agents of each type were trained on the reference problem with separate, random seeds with identical network parameters, hyperparameters, and training durations. The resulting training curves are shown in Figure 9. Notably, convergence behavior is broadly similar between each initialization within each agent category, which indicates that the spacecraft problem is well-posed and does not suffer from the same stochastic convergence that other common DRL environments produce. Clearly, the shielded agents produce substantially better mean rewards at virtually every point in the training process, with the final shielded agents achieving more than twice the mean reward of the unshielded agents. This performance is the result of two benefits of shielding: first, the shielded agents do not spend as much time exploring regions of the state/action space related to failure, as the shield activations keep the agent away from these regions; second, the "safety" aspect of the shield prevents the agent from receiving a reward penalty associated with failure. These results show that the addition of shielding to learning processes for typical spacecraft decision problems to which safety is a core attribute can dramatically improve performance even during training.

Performance Results

To verify that the agents are indeed performing in a safe manner, a simulator consisting of the agent in a closed-loop interaction with the environment was set up and run multiple times for the best-performing shielded and unshielded agents. The resulting phase-plot diagrams of the agent's behavior in the observed battery and wheel speed are demonstrated in Figure 10. The shielded learning agent is able to immediately recover after breaching the battery charge warning limit, and remains bounded by the wheel speed limit while converging to a limit cycle in the upper-right of the nominal section of the phase space.

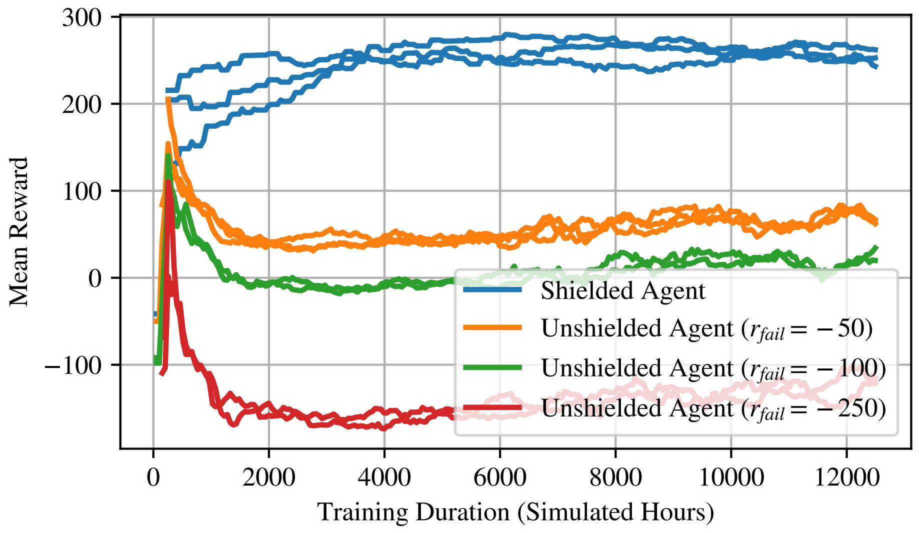
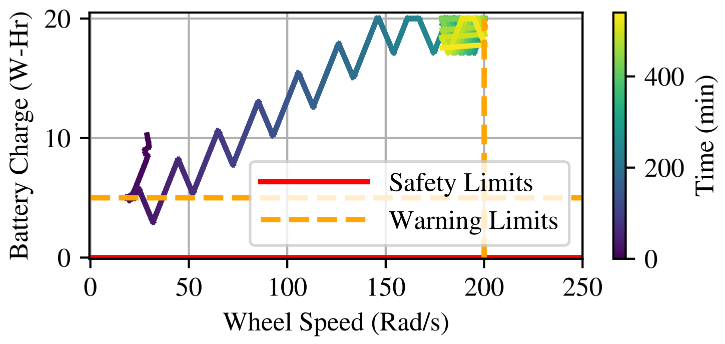


Figure 9: Comparison of achieved mean reward during training versus quantity of training time for shielded vs. reward-engineering approaches.

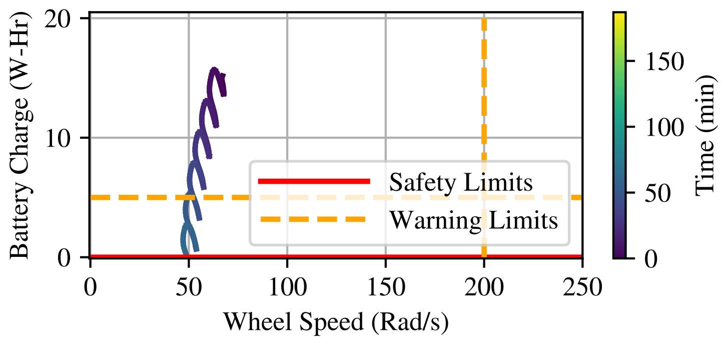
On the other hand, the unshielded agent allows itself to run out of power relatively quickly over the simulation period and does not recover, indicating that it has converged to a local minima in the training space.

Sensitivity to Model Errors

The use of simulated training data presents a risk of errors between modeled and real-life dynamics, which can lead to poor performance. To examine this, the trajectory analysis for both the shielded and unshielded agents demonstrated in Figure 10 was repeated over a range of spacecraft masses (which affects the spacecraft's inertia and therefore the required frequency of wheel desaturations) and power consumption levels. To provide a common figure of merit that captures both operational efficiency and safety, the reward function of Eqn. 9 is used for both agents, with a penalty of -1000 assigned for failure; this ensures that positive rewards always result in a positive score, while trajectories that fail always result in a negative score. The resulting reward plots are shown in Fig. 11. While some initial conditions always result in failure, these results show that the shielded agent performs relatively well across a range of environmental parameters, while the unshielded agent works well only in conditions that minimize the need to perform desaturation actions (i.e., when the spacecraft mass is large.)

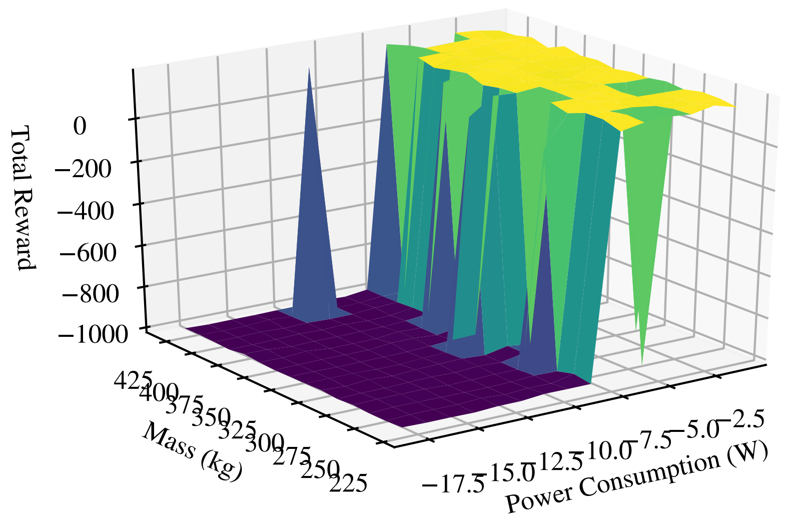


(a) Phase plot of the system observations for a run of the shielded agent.

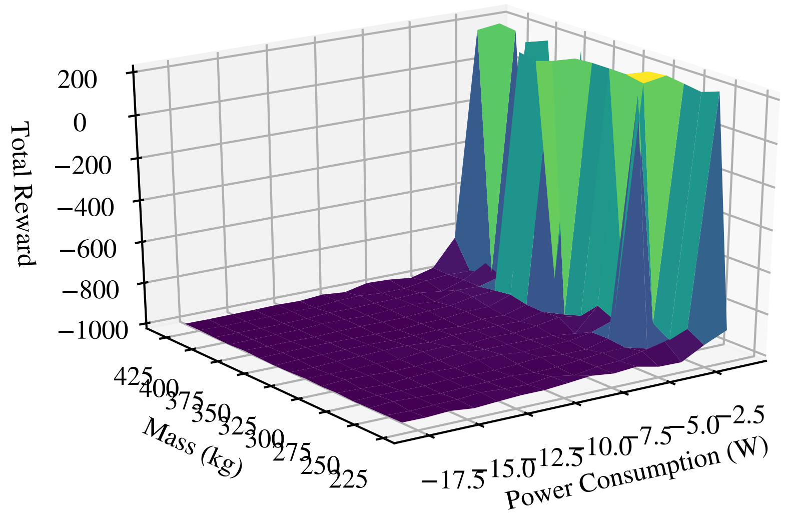


(b) Unshielded Agent; note that the agent fails by depleting the spacecraft's battery.

Figure 10: Observation phase plots for the shielded and unshielded agents.



(a) Shielded agent sensitivity



(b) Unshielded agent sensitivity

Figure 11: Reward sensitivity with respect to changing environment parameters.

CONCLUSION

A methodology for considering spacecraft command and control problems as sequential decision problems suitable for the application of modern machine learning tools has been presented and extended using the Basilisk astrodynamics framework. In addition, the technique of reactive synthesis and shielded reinforcement learning has been reviewed and applied to a detailed reference spacecraft command and control problem. In comparison to naive approaches to reinforcement learning, the shielded learning approach produces sequential decision agents that both operate safely under prescribed limits and achieves quantitatively better performance versus the unshielded learning agent.

REFERENCES

[1] John Alcorn, Hanspeter Schaub, Scott Piggott, and Daniel Kubitschek. Simulating Attitude Actuation Options Using the Basilisk Astrodynamics Software Architecture. *67 th International Astronautical Congress*, 2016.

[2] Mohammed Alshiekh, Roderick Bloem, Ruediger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe Reinforcement Learning via Shielding. *ArXiV*, pages 1–23, 2017.

[3] Anthony R Cassandra. A Survey of POMDP Applications. *Uncertainty in Artificial Intelligence*, pages 472–480, 1997.

[4] Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. PRISM-games: A model checker for stochastic multi-player games. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7795 LNCS:185–191, 2013.

[5] Steve Chien and Ari Jonsson. Automated Planning & Scheduling for Space Mission Operations JPL. (February), 2005.

[6] Steve Chien, Rob Sherwood, Daniel Tran, Rebecca Castano, Benjamin Cichy, Ashley Davies, Gregg Rabideau, Nghia Tang, Michael Burl, Dan Mandl, Stuart Frye, Jerry Hengemihle, Jeff D Agostino, Robert Bote, Bruce Trout, Seth Shulman, Stephen Ungar, Jim Van Gaasbeck, Darrell Boyer, Control Systems, Michael Griffin, and Hsiao-hua Burke Mit. Autonomous Science on the EO-1 Mission. *Proceedings of International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, (May), 2003.

[7] Steve A Chien, Daniel Tran, Gregg Rabideau, Steve R Schaffer, Dan Mandl, and Stuart Frye. Timeline-Based Space Operations Scheduling with External Constraints. *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, (Icaps):34–41, 2010.

[8] Teck H. Choo and Joseph P. Skura. SciBox: A software library for rapid development of science operation simulation, planning, and command tools. *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, 25(2):154–161, 2004.

[9] Alicia D Cianciolo, Robert W Maddock, Jill L Prince, Angela Bowes, Richard W Powell, Joseph P White, Robert Tolson, O Shaughnessy, and David Carrelli. Autonomous aerobraking development software : Phase 2 summary. pages 1–16, 2018.

[10] Cyrus Foster, Henry Hallam, and James Mason. Orbit determination and differential-drag control of Planet Labs cubesat constellations. *Advances in the Astronautical Sciences*, 156:645–657, 2016.

[11] C. R. Frost. Challenges and Opportunities for Autonomous Systems in Space. *National Academy of Engineering's U.S. Frontiers of Engineering Symposium*, 2010.

[12] Brian Gaudet, Roberto Furfaro, Markov Decision Process, Reinforcement Learning, Linear Quadratic Regulator, Tucson Arizona, and Tucson Arizona. Robust Spacecraft Hovering Near Small Bodies in. *test*, (August):1–20, 2012.

[13] Andrew Harris and Hanspeter Schaub. Towards Reinforcement Learning Techniques for Spacecraft Autonomy. *42nd Annual AAS Guidance, Navigation and Control Conference*, (AAS 18-078):1–10, 2018.

[14] Andrew Harris, Thibaud Teil, and Hanspeter Schaub. Spacecraft Decision-Making Autonomy Using Deep Reinforcement Learning. *29th AAS/AIAA Space Flight Mechanics Meeting, Hawaii*, (AAS 19-447):1–19, 2019.

[15] Kyle D Julian and Mykel J Kochenderfer. Autonomous Distributed Wildfire Surveillance using Deep Reinforcement Learning. (January):1–16, 2018.

[16] Daniel G. Kubitschek. Impactor Spacecraft Encounter Sequence Design for the Deep Impact Mission. *Jet Propulsion*, pages 1–14, 2005.

[17] Ilaria Bloise Roberto Furfaro. Deep Learning for Autonomous Lunar Landing. *Proceedings of the 2018 AAS/AIAA Astrodynamics Specialist Conference, Snowbird UT*, 2018.

[18] Eric Sample, Nisar Ahmed, and Mark Campbell. An Experimental Evaluation of Bayesian Soft Human Sensor Fusion in Robotic Systems. (August):1–19, 2012.

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *Arxiv*, pages 1–12, 2017.

[20] Oriol Vinyals. Deep Learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.