AAS 19-447

SPACECRAFT DECISION-MAKING AUTONOMY USING DEEP REINFORCEMENT LEARNING

Andrew Harris,[*] Thibaud Teil,[†] Hanspeter Schaub[‡]

The high cost of space mission operations has motivated several space agencies to prioritize the development of autonomous spacecraft control techniques. "Learning" agents present one manner in which autonomous spacecraft can adapt to changing hardware capabilities, environmental parameters, or mission objectives while minimizing dependence on ground intervention. This work considers the frameworks and tools of deep reinforcement learning to address high-level mission planning and decision-making problems for autonomous spacecraft, under the assumption that sub-problems have been addressed through design. Two representative problems reflecting challenges of autonomous orbit insertion and science operations planning, respectively, are presented as Partially-Observable Markov Decision Processes (POMDP) and addressed with Deep Reinforcement Learners to demonstrate the benefits, pitfalls, considerations inherent to this approach. Sensitivity to initial conditions and learning strategy are discussed and analyzed. Results from selected problems demonstrate the use of reinforcement learning to improve or fine-tune prior policies within a mode-oriented paradigm while maintaining robustness to uncertain environmental parameters.

INTRODUCTION

Spacecraft autonomy has long been regarded as a "holy grail" of spacecraft guidance, navigation, and control research.[1] Decades of development have yielded few fully autonomous spacecraft; instead, mission planners and operators increasingly rely on automated planning tools to inform and support high-level decision making, in part due to the ability of human operators to act under environmental uncertainty, changing or competing mission objectives, and hardware failure. Advances in the field of artificial intelligence and machine learning techniques present one avenue in which these problems can be addressed without bringing humans into the loop. This work aims to extend the state-of-the-art in spacecraft autonomy by applying contemporary machine learning techniques to the high-level mission planning problem.

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

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

[‡]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.

At present, examples of spacecraft autonomy typically fall into two categories: rule-based autonomy and optimization-based autonomy. Rule-based autonomy treats a spacecraft as a state machine consisting of a set of mode behaviors and defined transitions between modes. Pioneered by missions like Deep Impact,[2] and currently used by missions such as the PlanetLabs constellation,[3] spacecraft using rule-based autonomy transition between operational and health-keeping modes (charging, momentum-exchange device desaturation) autonomously without ground contact. Typically, the design of these autonomous mode sequences is pre-defined or based on pre-launch criteria. Rule-based approaches are attractive from an implementation perspective, as they require little computing power and can be tested to validate their rigid transition criteria. Nevertheless, these rigid rule-sets are "brittle" to changes in mission parameters, such as hardware failures or new science objectives. They also require accurate understanding of the mission environment in order to prepare for unwanted behavior. Additionally, rule-based approaches do not readily support the integration of multiple competing mission objectives, and require that those trades be made on the ground with humans in the loop before mission sequences are uploaded.

In contrast to rule-based approaches are a class of tools that use models of spacecraft behavior and hardware to generate mission plans on the ground while considering mission objectives, which this work broadly describes as "optimization-based" autonomy. Within this class of algorithms, the spacecraft and its mission are viewed 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 that precludes their use on-board. This method also requires realistic models, and well developed testing environments. 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.[4]

As both rule- and optimization-based autonomy techniques become more mature, the search for "next-generation" spacecraft autonomy approaches has begun. As with contemporary approaches, emerging techniques in autonomy should reduce mission development and operational cost while improving mission returns. New techniques should also improve upon autonomy runtime, ability to deal with uncertainty, ability to learn from past experiences, and the ability to make mission-level decisions. The need for adaptability strongly suggests the use of machine learning (ML) techniques as a core of autonomous decision software.

Recent advances in machine learning may hold the key to these next-generation approaches for spacecraft autonomy and on-board decision-making, as they by definition allow agents to improve their behaviors as they gain experience. Contemporary reinforcement learning approaches, for example, do not require knowledge of system models and scale relatively well to large problems with multiple constraints or non-convex reward functions.[5] This work aims to explore the applications and frameworks necessary to apply deep

reinforcement learning to the spacecraft decision-making problem.

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 6 and 7 have considered reinforcement learning in the context of autonomous aerobraking planners, with mixed results. Others explore machine learning techniques for asteroid proximity operations[8] or autonomous lunar landing.[9] Importantly, these approaches have focused on low-level control with reinforcement learning, an area that has been traditionally been covered by conventional estimation and control techniques with great success. In contrast, this work explicitly examines applications of reinforcement learning to high-level spacecraft planning and decision-making problems that have traditionally been the domain of rigid expert policies or optimization-focused strategies.

This work explores an approach to fit spacecraft autonomy into the Partially-Observable Markov Decision Process (POMDP) framework and its general solution through model-free "Deep-Q" reinforcement learning. First, background on POMDPs and RL are presented and their relevance to the spacecraft autonomy problem are outlined. Next, several demonstration problems representative of applications for spacecraft autonomy are presented within this framework, and solved using Deep-Q reinforcement learning. A brief overview of verification strategies for this approach are provided. Finally, these results are discussed and future work within the field outlined.

PROBLEM STATEMENT

In the search for future autonomy approaches, it is desirable to both replicate existing capabilities in the realm of rule-based and optimization-oriented autonomy while improving their extensibility, robustness to unmodeled 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 "solving" such problems are POMDPs compactly represent the processes facing a software agent acting in an evolving environment according to some higher-level objective.[10] 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 Figure 1, and discussed further here. As in traditional Markov Decision Processes, 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):

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

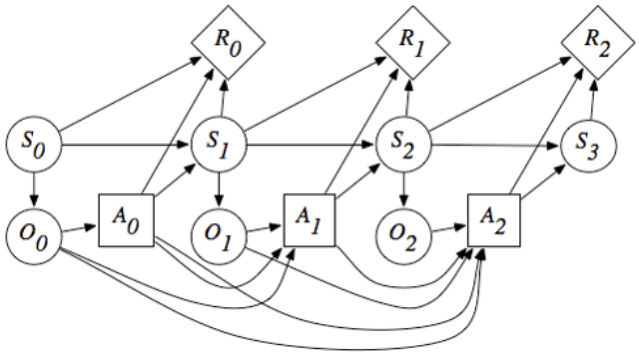


Figure 1. Partially Observable Markov Decision process framework for considering decision problems.

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:

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

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[11] or multi-vehicle coordination problems.[12]

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.

Mode-Based Planning

A long-standing problem in the application of reinforcement learning is the manner in which prior knowledge about a given system can be applied.[5] For space applications, many

"low-level" control and estimation problems are well-solved by traditional techniques; rather than training a learner to re-implement orbital mechanics, it is preferable to leverage existing domain knowledge and shift the application of autonomy algorithms to high-level mission planning. In effect, this approach breaks the spacecraft command problem into a set of sub-problems that are assumed to be addressed using traditional approaches, allowing the autonomous agent to focus on the proper scheduling and sequencing of these software "modes" to meet mission objectives.

Mission sequences are discretized in time and dynamics by the application of spacecraft operational "modes," which describe discrete classes of spacecraft states. Modes occur for finite durations reflecting the needs of the behavior compartmentalized by said mode; for example, a "control" mode would be designated to run for the settling time of the control system before handing off command of the spacecraft systems back to the planning agent.

This approach has a number of benefits. First, the specific behaviors of individual modes can be abstracted away and solved using prior knowledge and traditional techniques, allowing for existing knowledge about spacecraft control or mission needs to be applied. At the same time, this approach reduces the action space for a planner from an infinite-dimensional space of control inputs to a set of discrete modes, thereby reducing computational burden. Finally, this action realization does not preclude the use of continuous state, observation, or reward spaces.

Reinforcement Learning Solutions

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 techniques—a subtype of machine learning which focuses on deriving solutions to POMDPs from prior experiences—are not restricted to addressing problems with continuous models, or indeed with any models at all.

Traditionally considered in the context of discretized systems ("tabular" RL), recent advancements in the training of large Artificial Neural Networks (ANNs) has led to the development of so-called "Deep" RL techniques.[13] In comparison to traditional tabular reinforcement learning, Deep RL can directly consider infinite-dimensional input and output spaces without the need for discretization, and without losing the ability to address systems consisting of both discrete and continuous states and actions. A suite of representative Deep Learners has been implemented in Python for use with OpenAI's `gym`[14] environment, which provides a standard interface for reinforcement learning problems. A portion of this work exploresthe performance of different model-free Deep Reinforcement learners with respect to end performance, computational cost, and data efficiency.

Reinforcement learning offers a variety of approaches to solving general Markov De-

cision 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 function is discounted sum of the rewards from the current time onward 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.

Deep Q-Learning

Deep Q-Learning (DQN) performs Q-Learning updates, but the action-value function, Q, is learned using a fully connected neural network. The neural network consists of an input layer with a neuron for each agent state in s. The output layer contains a neuron for each possible action, a. This implementation produces a Q value for each possible action given agent's current state. The loss function for the Q-Learning neural network (Q-network) is

$$L_k(\theta_k) = \left(r + \gamma \max_{a'} Q(s', a'; \theta_k) - Q(s, a; \theta_k)\right)^2 \tag{8}$$

where θ_k are the Q-network parameters at iteration k, and $Q(s', a'; \theta_k)$ represents the Q-function evaluated for s' and a' with the parameter set θ_k. RMS error is used as the loss function for back-propagation. Instead of updating a table with new values of Q, the DQN algorithm performs back-propagation on the Q-network with a new target value.

Recent research has produced a number of tweaks and incremental improvements to DQN implementation that improves its convergence properties and stability. The DQN algorithm used to demonstrate our results includes the following architectures and improvements:

1. Epsilon Annealing: The explore-exploit trade-off inherent to reinforcement learning is dealt with here by using an epsilon-greedy strategy, wherein actions are selected according to the agent's understanding of maximum value with probability $1 - \epsilon$ and randomly otherwise. Here, ϵ is reduced linearly from 1 for a portion of the training period to drive convergence.[5]

2. Target Network: DQN reward progress is often unstable during training. The agent may look like it has converged, but in another few episodes, will appear to have forgotten all of its training. Target networks help to prevent this phenomenon by decoupling the feedback in Bellman's equation in Eq. (9). The target values for the Q network update now use a frozen Q network that is update periodically (every C steps in algorithm 1). This change increases network convergence stability during training.[15] This change results in a loss function that is updated using the target network's parameters, $\hat{\theta}_k$:

$$L_k(\theta_k) = \left(r + \gamma \max_{a'} Q(s', a'; \hat{\theta}_k) - Q(s, a; \theta_k)\right)^2 \tag{9}$$

3. SARSA (State-Action-Reward-State-Action): This is an adjustment to the loss function such that the value function does not always select the next maximum-value action, but instead uses the experienced action transitions to update the reward function to prevent "burn-in" behaviors from bad random seeds.[5] After this change, the loss function is written as

$$L_k(\theta_k) = \left(r + \gamma Q(s', a'; \hat{\theta}_k) - Q(s, a; \theta_k)\right)^2 \tag{10}$$

4. Eligibility Trace: Sample sparsity is an issue for space systems; additionally, multi-step behaviors may be missed by a simple one-step Bellman update. To this end, additional future state-action pairs are sampled using an eligibility trace methodology, which adds additional discounted state-action pairs sampled along the same trajectory to the current update. This n-step backup adjusts the loss function further to

$$L_k(\theta_k) = \left((r + \sum_{i=1}^{n} \gamma^i Q(s^i, a^i; \hat{\theta}_k) + -Q(s, a; \theta_k)\right)^2 \tag{11}$$

5. Replay Memory: Recency bias is an issue when rewards are sparse or when optimal behavior is unlikely to come out of random action sequences. To account for this, the DQN algorithm used in this paper (algorithm 1) utilizes a replay memory. A finite backlog of agent experiences are 'remembered' after each environment step. The replay memory is sampled at each step to train to the Q network. Replay memory enables the agent to train on successful experiences multiple times and helps to prevent 'catastrophic forgetting'.[15]

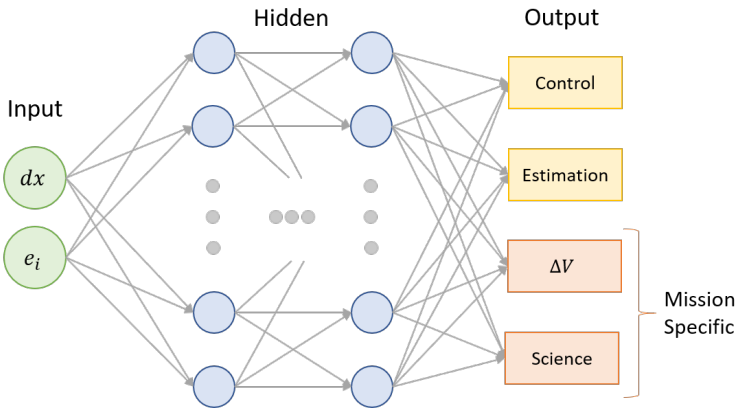


Figure 2. Q Network Structure for Station Keeping and Mars Orbit Insertion.

This algorithm is deeply sensitive to random seed, hyperparameters (i.e., parameters that inform and guide the parameter learning, such as the learning rate or number of neurons), and the reward function.[16] It is often difficult to predict whether or not the algorithm will converge to a reasonable policy. As a result, there are numerous modifications to the traditional DQN algorithm to increase the stability of learning and to ensure convergence for a variety of parameters.

REPRESENTATIVE SCENARIOS

While Deep-Q Networks and Partially-Observable Markov Decision processes are general enough to be applied to virtually any problem, implementation-specific details can be sufficient to prevent learner convergence or harm performance. For this reason, a pair of scenarios reflecting challenges faced by an autonomous spacecraft attempting an interplanetary mission are presented here as POMDPs. Both scenarios share common dynamics models and several common actions, and can therefore be taken to represent the same spacecraft at different points in its mission life-cycle.

The "true" non-linear dynamics resulting from gravity interactions are taken to follow the two-body equations of motion in the presence of perturbing accelerations:

$$\ddot{\boldsymbol{r}} = \frac{-\mu}{r^3}\boldsymbol{r} + \boldsymbol{a}_p \tag{12}$$

At the same time, a pre-defined reference trajectory obeying two-body dynamics without perturbing accelerations is used to define the desired mission:

$$\ddot{\boldsymbol{r}}^* = \frac{-\mu}{r^{*3}}\boldsymbol{r}^* \tag{13}$$

The ignorant propagator in Equation (13) is also used to propagate forward the spacecraft's current orbital state estimate, $\hat{x}$. The resulting state and estimate errors are defined as

$$\boldsymbol{e}_s = \boldsymbol{x} - \boldsymbol{x}^*, \ \boldsymbol{e}_{\text{est}} = \boldsymbol{x} - \hat{\boldsymbol{x}} \tag{14}$$

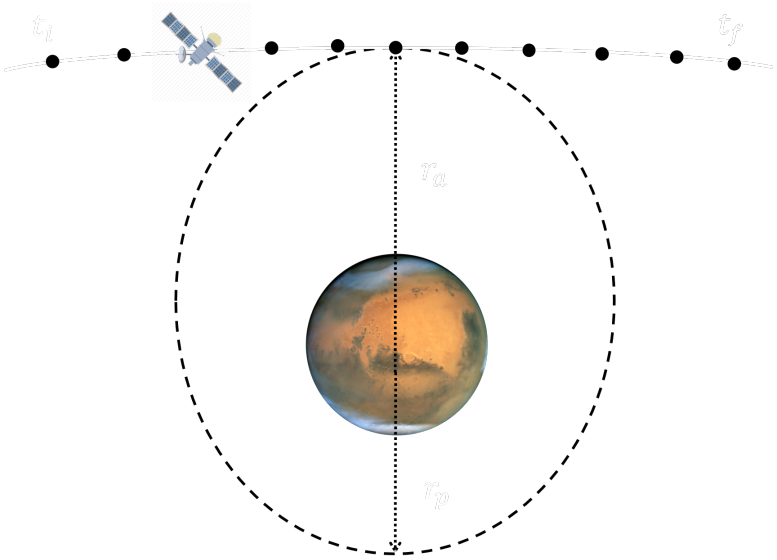


Figure 3. Visual depiction of the Mars Orbit Insertion scenario.

Likewise, the spacecraft-internal control error is defined as:

$$\boldsymbol{e}_c = \hat{\boldsymbol{x}} - \boldsymbol{x}^* \tag{15}$$

It is assumed that a team of guidance and controls engineers has devised a pair of mutually-exclusive software states that cause the spacecraft's estimation and control errors to decay exponentially. A necessary task for both of the presented scenarios therefore includes the management of estimate and control error while accomplishing other mission objectives. Under the mode-based planning paradigm, the planner considers the state and error dynamics over discrete time-steps that are scenario specific.

These models, alongside a family of deep reinforcement learning agents, is implemented in Python using the OpenAI `gym` framework[14] to represent the spacecraft-mode POMDP interface in a standardized manner. The deep learning agents are created using Keras [*] using Tensorflow[17] back-end.

Orbit Insertion

The first environment simulates insertion into orbit about a planet, as is represented visually by Figure 3. This is an example of an interplanetary mission in which a spacecraft is flying towards Mars, and needs to conduct an impulsive maneuver at the correct time to enter orbit about Mars under uncertain knowledge of its state. This type of decision is crucial to the success of many interplanetary missions, and therefore represents an important challenge for proposed autonomy systems. This is also an example of a timely maneuver in which any unexpected behavior could be lethal to the mission as ground teams could

[*] `https://keras.io`

Table 1. Initial Conditions Dispersons for Orbit Insertion

Parameter	Dispersion
Semi-major axis of true trajectory	$\mathcal{N}$(-2000, 100)
Eccentricity of true trajectory	$\mathcal{N}$(2, 0.01)
True anomaly of true trajectory	$\mathcal{N}$(-1.5, 0.01)
Semi-major axis of estimated trajectory	$\mathcal{N}$(-2000, 100)
Eccentricity of estimated trajectory	$\mathcal{N}$(2, 0.01)
True anomaly of estimated trajectory	$\mathcal{N}$(-1.5, 0.01)

not react in time. All simulations are run with an unmodeled acceleration of Mars' second gravitational spherical harmonic J_2 for true propagation.

Under the mode-control paradigm, the spacecraft considers an additional mode, "thrust," which applies an impulsive ΔV computed with it's estimated states at a specific time and reflects a major maneuver to adjust its trajectory. The challenge of the orbit insertion scenario is therefore to ensure that this thrust is applied at the correct time to ensure orbit insertion, while at the same time maintaining accurate position estimates and controlling towards a defined reference trajectory. Furthermore, the knowledge of the true spacecraft states are paramount to the spacecraft thrusting correctly.

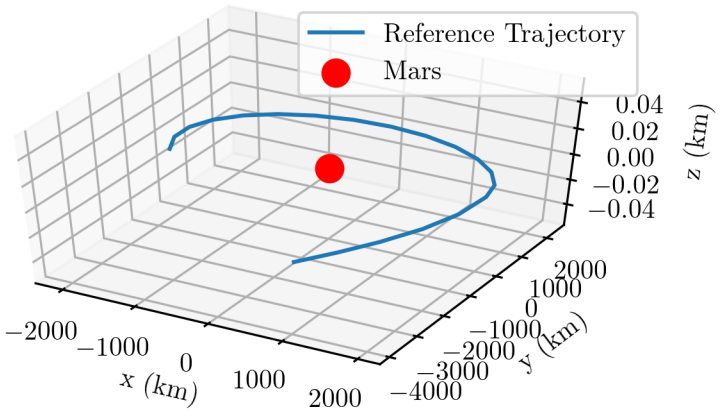


Figure 4. Reference Trajectory changed by thrust maneuver

Figure 4 pictures the reference trajectory. The initial orbit is a hyperbolic fly-by with a semi-major axis of $a = -2,000$km, eccentricity of $e = 2$, and initial true anomaly of $\nu = -1.5$ rad. The goal orbit has a semi-major axis of $a = 2,000$km, eccentricity of $e = 0.01$, inclination of 0rad, true anomaly of $\nu = 0.01$ rad. This scenario therefore brings a spacecraft from a hyperbolic fly-by orbit into a captured circular orbit around it's target planet (Mars). Each mode has a length of 2 minutes with a time step of 5 seconds.

Table 1 shows the dispersions on the initial conditions that are used for training in the section on MOI results.

Science Station Keeping

After orbit insertion, spacecraft typically transfer to a "mission" orbit and begin conducting operations. However, while in this orbit, un-modeled perturbations will steady force the spacecraft away from its desired orbit, thereby degrading or preventing mission operations from taking place.

This environment approximates this behavior with the Estimation and Control modes described above with the addition of a reward-producing "science" mode that reflects mission-oriented behaviors. Within the presented framework, activating the Science mode causes the spacecraft to achieve a reward proportional to its distance from the target orbit. To represent attitude or instrument constraints, it is assumed that the spacecraft state estimate and the true spacecraft state drift away from the reference in this mode.

Ten individual modes are considered, approximating one orbit's worth of behavior.

SCENARIO RESULTS

Science Station Keeping

The science station-keeping problem is attractive for the aforementioned framework and serves as a representative example for a simple mode-scheduling problem. Initially, a simple Deep-Q network was implemented to control the system. The network inputs were taken to be the reference position and velocity, the estimated position and velocity, and the estimated position and velocity covariances to represent the spacecraft's knowledge of uncertainty. Combined, this represented an 18-state observation vector at the end of each mode. Reward was provided in a sparse manner based on the actions taken:

$$r_{\text{sci}} = \frac{r_{\text{base}}}{(1 + \boldsymbol{e}_c^T \boldsymbol{e}_c)} \tag{16}$$

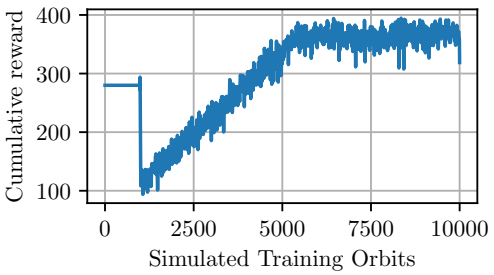
which ensures that the reward in science mode is always greater than or equal to zero, but still drops off as the orbit error increases. Reward in other modes is zero, reflecting their lack of contribution to the mission objective. Additionally, by ensuring that the reward is always positive, this process simplifies troubleshooting the training of the Q-approximating neural network and avoids large swings in the value of Q depending on whether or not science is performed.

In the hope of improving performance, an "expert-designed" prior policy is used to initialize the neural network before engaging in random exploration/exploitation. Unfortunately, this "naive" approach to deep-Q learning frequently yielded non-convergent results, as shown by the reward versus training episode plot in Figure **??**.

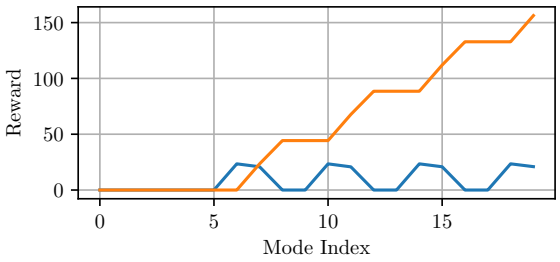
To remedy this, the neural network inputs are simplified. Rather than including all 18 desired, estimated, and covariance states, the learner is provided with the 2-norm of the estimated control error and the 2-norm of the diagonals of the covariance matrix.

$$O = \{\delta x = ||\hat{\boldsymbol{x}} - \boldsymbol{x}^*||, C = ||\text{diag}(C)||\} \tag{17}$$

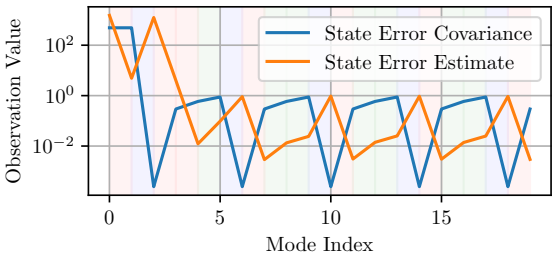
This has two effects: first, it simplifies the observation from eighteen states to two, which reduces the number of parameters the network needs to train on; second, it directly relates the input parameters to the designated control modes, aiding linear separability (which is thought to improve RL performance).[16] Under this modification, performance of the learner dramatically improved in terms of both training time and received reward. Under the final hyperparameters listed in Table 2, as well as the addition of deep eligibility traces to the learner, the positive results of Figure 5 are generated. As shown in Figure 5, the converged learner improves upon the expert prior policy by a factor of 1.5×, and successfully learns to estimate and control before entering a science mode.



(a) Reward as a function of episode during training



(b) Reward over the final simulated orbit period. Blue bars represent per-mode control, while orange represents the cumulative reward



(c) Mode sequence produced by the converged algorithm. Lines represent observed covariance (blue), and control error (red). Blue bars indicate an estimator mode, red bars indicate a control mode, and green bars indicate a science mode

Figure 5. Converged station keeping results

Orbit Insertion

A second Deep-Q learning agent is implemented to address the orbit insertion scenario. Orbit insertion introduces new challenges to overcome in order to robustly train agents for autonomous decision-making. These challenges revolve around the binary success criteria of the thrust maneuver. The ability to thrust is an action of equal weight to estimation or control modes. This encourages the agent in training to use it promptly, and very frequently

Table 2. Hyperparameters used in training the final StationKeep iteration.

Parameter	Value/Type
Number of Hidden Layers	1
Hidden Layer Depth	64
Hidden Layer Activation	ReLU
Output Layer Activation	Linear
Science Reward Multiplier	1
Learning Rate	0.01
Number of Training Episodes	10,000
Annealing Segment Length	3,000

fall into a local minimum that consists in thrusting immediately and attempting to minimize further errors with the control mode.

These challenges are overcome by implementing an eligibility trace and by forcing the thrust maneuver to occur in a 5-step window, centered around the expected thrust time (8th step given the initial conditions). This gives the RL agent the ability to optimize the thrust time, but doesn't require it to learn known astrodynamics facts: that the thrust-time is optimal at periapse of the hyperbolic orbit. By including knowledge of the dynamic models, the Deep-Q learning paradigm can be augmented to accommodate for an otherwise difficult task to optimize around. Furthermore, the reward function is smoothed around the optimal thrust time to help the optimization process to navigate the reward surface without discontinuities. The reward function in this scenario is less sparse than previously. A term is added for the state error (estimated state with respect to the reference), another term is added for control cost, and a positive reward is added for thrust timing:

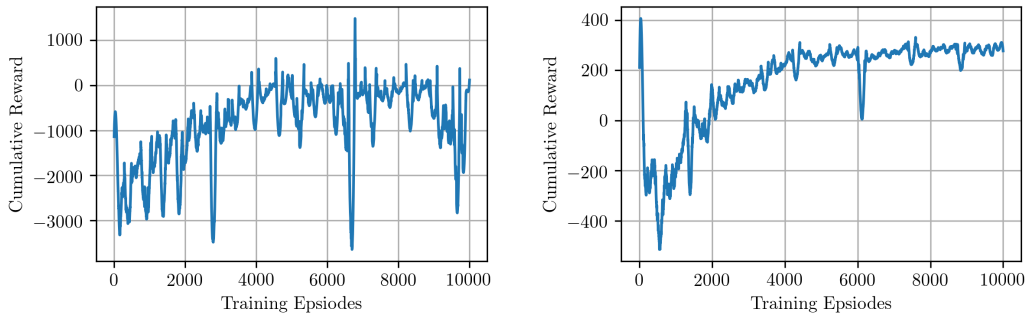
$$r(\boldsymbol{e}, c, \Delta\mathrm{V}) = \underbrace{w_e \boldsymbol{e}_c^T \boldsymbol{e}_c}_{\text{state error}} + \overbrace{w_c c}^{\text{control cost}} + \underbrace{w_t \Delta\mathrm{V}}_{\text{thrust reward}} \tag{18}$$

Where $\boldsymbol{e}$ is the state error, c is the control authority used and ΔV is the norm of the thrust impulse. w_e, w_c, and w_t are the weights corresponding to the state error, the control cost, and the thrust respectively. The values of these weights are a key component to the success of the agents learning. If the weight on the state error is large relative to the other costs, a poorly timed thrust could make that penalty abruptly skyrocket. However, if it is too small (or not accounted for) the spacecraft will not continue controlling orbit position after the thrust. Furthermore, it is difficult for the agent to learn to be on the correct trajectory in order to thrust at the right time and position.

Weights that allow for successful training are listed in Table 3, where t_{thrust} is the time at which the thrust action is taken by the agent, while t_{ref} is the time at which the reference changes orbits. This form for w_t allows for the aforementioned smoother, more continuous decay in the reward for thrusting off target. The scale of 100 is for the reward to be commensurate with the state errors penalties.

Table 3. Weights in the Orbit Insertion Reward Function

Weight	Value
w_e	-0.1
w_c	-0.05
w_t	$\frac{100}{1+\lvert t_{\text{thrust}} - t_{\text{ref}} \rvert}$



(a) Smoothed reward during deterministic training

(b) Smoothed reward during dispersed training

Figure 6. MOI Reward as a function of episode in training

In order to further help with these challenges, a rational agent is also run for 100 episodes of the training. This rational agent alternates between estimation and control, and thrusts at the 8th step which is exactly when the reference changes. As in the stationkeeping environment, this helps the agent in the initial exploration phase by guaranteeing the discovery of the global minimum. Hyperperameters used in the Orbit Insertion scenario are listed in Table 2. The network inputs are the spacecraft state error with respect to the reference trajectory, the norm of the estimator covariance of these states, and the time to expected thrust. The first two components of the network inputs are identical to the functioning station-keeping environments. The time to expected thrust transmits the knowledge of the simulation time relative to the expected thrust maneuver to the network.

The agent is trained in the MOI environment with deterministic initial conditions and random initial conditions. Figure 6(a) shows the results of the RL algorithms reward as a function of episode of training in the deterministic case. The rational agent appears to be optimal—with cumulative rewards of 331—and the trained agent is able to replicate the correct maneuver. Figure 6(b) shows the same reward plot with dispersed initial conditions for the training. In the random case, the rational agent performs far less well with cumulative rewards averaging around -800. The trained agent outperforms the rational agent in the end of it's training with usually positive rewards sometimes reaching 300.

The orbit insertion environment posed challenges usually tied to the discrete nature of the thrust command. Yet with some knowledge of the model and by preventing clearly undesirable actions, a RL agent is able to optimize the necessary actions within 10,000 runs. Figure 7 shows the state and estimation errors and thrust maneuver for one of the

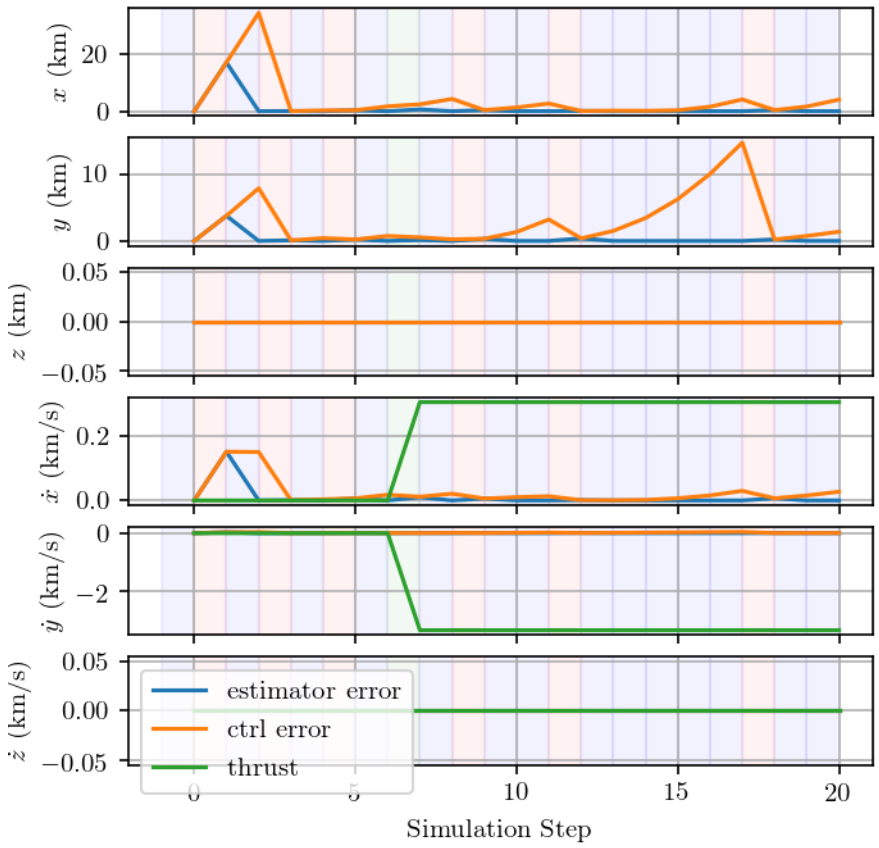


Figure 7. Mode Sequence for dispersed Orbit Insertion

dispersed runs. As could be expected, the agent first estimates to know its state error, then controls to the reference before thrusting at the correct step (8th step). Then it continues to alternate between estimation and control after the thrust in order to minimize reward losses. The thrust line shows the value of the thrust vector that is applied during the maneuver. The three trajectories are pictured in Figure 8.

AGENT VALIDATION AND GENERALIZATION

On-board applications of autonomy for spacecraft require not only a high level of performance, but also the ability to be rigorously validated. Additionally, overfitting of deep reinforcement learning agents frequently prevents DRL agents from applying their knowledge successfully when environment parameters change, thereby undercutting one of the primary advantages of reinforcement learning as a basis for autonomy. While rigorous proofs of performance are not yet available for deep reinforcement learning, the mode-based framework allows for the use of tools from hybrid systems to validate trained deep

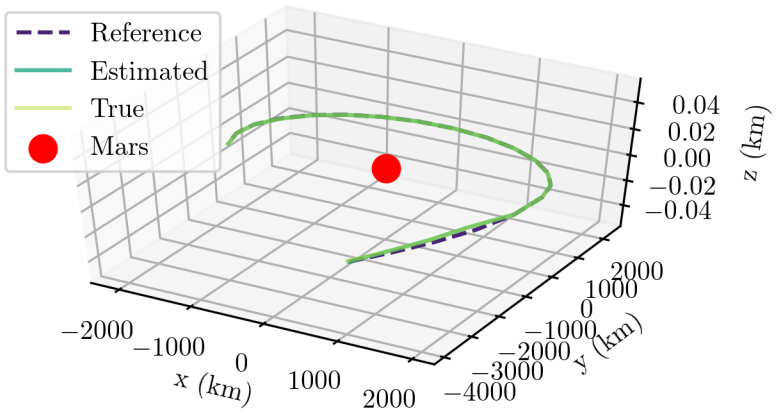


Figure 8. Three trajectories during insertion maneuver

reinforcement learning agents. The described mode-based framework directly falls into the category of continuous-time switched-mode systems (CSMS), which is stated formally as a system whose dynamics obey:

$$\dot{\boldsymbol{x}} = f_i(\boldsymbol{x}(t)) \; \forall \; x(t) \in X_i \tag{19}$$

where X_i represents the state space of the switched system and i represents one of the system's switched dynamics. Branicky[18] describes the necessary conditions for such a system to be stable in the sense of Lyapunov using the theory of multiple Lypaunov functions. One notion of stability that arises from this definition is the convergence of a switched system to a limit set or limit cycle. Figure 5(c) shows that the trained agent appears to produce cyclic behavior in the observed parameters. This suggests that the system's multiple candidate Lyapunov functions can be taken directly from the environment's observation function:

$$V_1 = \delta \boldsymbol{x} \tag{20}$$

$$V_2 = ||\text{diag}(C)|| \tag{21}$$

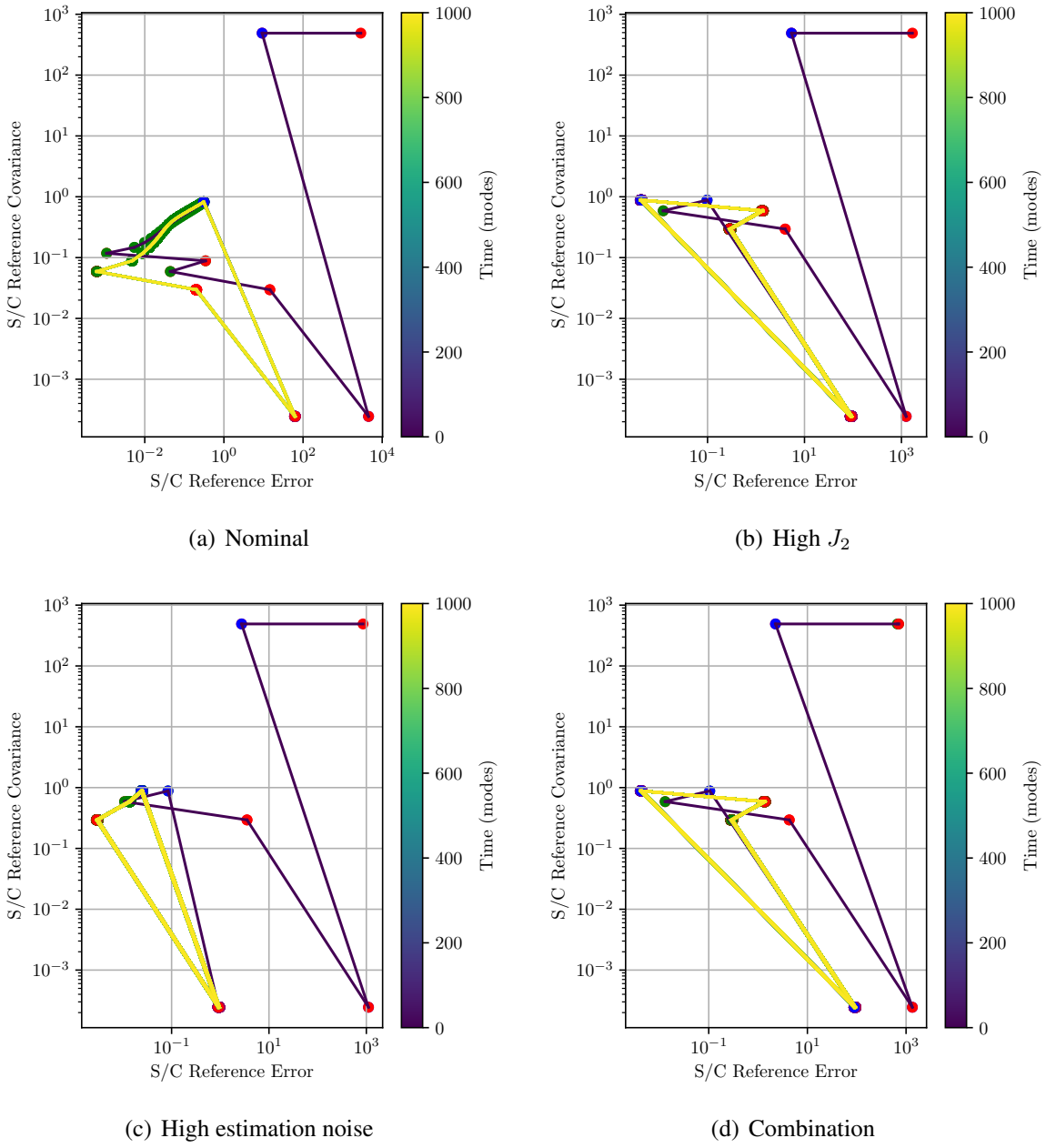
To evaluate the trained agent's convergence properties, these functions are observed over a lengthened simulation run and plotted against each-other in the style of a phase-plane. Clearly, the agent is able to guide the mode-switching process not only to maximize the achieved reward, but also to stabilize the system towards a limit cycle while doing so.

With this testing approach established, the sensitivity of the trained agent to changes in the environment. Three cases are evaluated, as listed in Table 4; these cases are intended to showcase the effects of changing environmental disturbance magnitudes, worsening sensor performance, and the combination of these two attributes relative to the training environment. The results of this test are shown in Figure 9.

None of the tested cases produced divergent behavior even with large changes to the environment's parameters. While the shape of the limit cycle in the phase plane appears to

Table 4. Environmental parameters used to validate the Science Stationkeeping environment.

Value	Nominal	High J_2	High Sensor Error	Combination
J_2	J_2	100 J_2	J_2	100 J_2
λ	0.01	0.01	1	1



(a) Nominal

(b) High J_2

(c) High estimation noise

(d) Combination

Figure 9. Phase plots demonstrating stationkeeping convergence. Dots represent mode transitions (red=control, blue=estimation, green=science) while joining line color represents time.

be environment-specific, convergence is clearly achieved to this cycle without modification to the agent itself. Notably, due to the reward structure of the environment, many of the high-noise cases do not result in large rewards; however, this can be interpreted as the agent attempting to jockey for a position in which rewards might be available. As these behaviors occurred without retraining on a different timescale, they are taken to represent generalization with respect to the environment's parameters.

CONCLUSION

This work has successfully reformulated the general spacecraft decision problem into the POMDP framework, setting the stage for future studies in this area through an open-source library using OpenAI's `gym` framework. The use of reinforcement learning techniques has been adapted to the spacecraft state machine paradigm: combining Deep-Q learning with POMDP to produce policies that are comparable to "expert" priors.

Various reward structures, hyperparameters, and environment parameters have been considered throughout the experimentation. It appears that failure to obtain positive results in applying Deep-Q reinforcement learning to the spacecraft control problem is the result of insufficient sampling of the problem space. This issue is exacerbated by the large computational requirements for the environments, which slows the training problem considerably compared to simpler environments.

Additionally, the mode-based paradigm for designing future decisionmaking algorithms directly lends itself to verification by way of hybrid systems theory. This work presents one optic by which that theory could be used to define "successful" or "stable" autonomous decision agents.

Future work includes the investigation of model-based reinforcement learning techniques to decrease sample requirements and leverage existing knowledge about the space environment. Additionally, higher-speed models built using the *Basilisk* astrodynamics framework[†] will be investigated to speed training time. Furthermore, in order to further improve the orbit insertion performance, it could be judicious to work back up from a functional station keeping environment. By adding the thrust command with high delta-V, and by restricting the authority of the control action, the learner will need to use this action to minimize costs. Adding this and progressively adding complexity the scenario could yield more consistent results.

REFERENCES

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

[2] D. G. Kubitschek, "Impactor Spacecraft Encounter Sequence Design for the Deep Impact Mission," *Jet Propulsion*, 2005, pp. 1–14.

[3] C. Foster, H. Hallam, and J. Mason, "Orbit determination and differential-drag control of Planet Labs cubesat constellations," *Advances in the Astronautical Sciences*, Vol. 156, 2016, pp. 645–657.

[†] `http://hanspeterschaub.info/bskMain.html`

[4] S. A. Chien, D. Tran, G. Rabideau, S. R. Schaffer, D. Mandl, and S. Frye, "Timeline-Based Space Operations Scheduling with External Constraints," *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, No. Icaps, 2010, pp. 34–41.

[5] R. S. Sutton and A. G. Barto, "Reinforcement learning," *Learning*, Vol. 3, No. 9, 2012, p. 322, 10.1109/MED.2013.6608833.

[6] A. Harris, "Towards Reinforcement Learning Techniques For Spacecraft Autonomy," *AAS Guidance, Navigation and Control Meeting*, 2018, pp. 1–10.

[7] A. D. Cianciolo, R. W. Maddock, J. L. Prince, A. Bowes, R. W. Powell, J. P. White, R. Tolson, O. Shaughnessy, and D. Carrelli, "Autonomous Aerobraking Development Software : Phase 2 Summary," *2013 AAS/AIAA Astrodynamics Specialist Conference*, 2018, pp. 1–16.

[8] B. Gaudet and R. Furfaro, "Robust Spacecraft Hovering Near Small Bodies in Environments with Unknown Dynamics Using Reinforcement Learning," *2012 AIAA/AAS Astrodynamics Specialist Conference*, No. August, 2012, pp. 1–20, 10.2514/6.2012-5072.

[9] I. B. Roberto Furfaro, "Deep Learning for Autonomous Lunar Landing," *Proceedings of the 2018 AAS/AIAA Astrodynamics Specialist Conference, Snowbird UT*, 2018.

[10] A. R. Cassandra, "A Survey of POMDP Applications," *Uncertainty in Artificial Intelligence*, 1997, pp. 472–480.

[11] K. D. Julian and M. J. Kochenderfer, "Autonomous Distributed Wildfire Surveillance using Deep Reinforcement Learning," No. January, 2018, pp. 1–16, 10.2514/6.2018-1589.

[12] E. Sample, N. Ahmed, and M. Campbell, "An Experimental Evaluation of Bayesian Soft Human Sensor Fusion in Robotic Systems," No. August, 2012, pp. 1–19, 10.2514/6.2012-4542.

[13] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, Vol. 521, No. 7553, 2015, pp. 436–444, 10.1038/nature14539.

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016, pp. 1–4, 10.1021/am3026129.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, No. 7540, 2015, p. 529.

[16] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," 2017.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. Software available from tensorflow.org.

[18] M. S. Branicky, "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems," *IEEE Transactions on Automatic Control*, Vol. 43, No. 4, 1998, pp. 475–482, 10.1109/9.664150.