Adaptive Continuous Control of Spacecraft Attitude Using Deep Reinforcement Learning

Conference Paper · August 2020

CITATIONS
4

READS
2,371

3 authors:

Jacob Elkins
University of Alabama in Huntsville
3 PUBLICATIONS **10** CITATIONS
SEE PROFILE

Rohan Sood
University of Alabama
67 PUBLICATIONS **466** CITATIONS
SEE PROFILE

Clemens Rumpf
NASA
42 PUBLICATIONS **181** CITATIONS
SEE PROFILE

AAS 20-475

ADAPTIVE CONTINUOUS CONTROL OF SPACECRAFT ATTITUDE USING DEEP REINFORCEMENT LEARNING

Jacob G. Elkins[*], Rohan Sood[†], and Clemens Rumpf[‡]

As modern and future space missions plan to explore diverse bodies across deep space, the ability to conduct successful spacecraft operations under uncertainty becomes increasingly apparent. A highly accurate and precise spacecraft attitude controller that is robust to the perturbation forces encountered in an uncertain dynamics environment is critical to the ongoing success of spaceflight in deep space. In this work, we present a framework for deriving an adaptive spacecraft attitude controller using deep reinforcement learning. The controller developed is shown to effectively perform large-angle slew maneuvers at industry-standard pointing accuracies. We find that the controller is capable of adapting in the presence of various disturbance torques unseen during training and is system-agnostic of the spacecraft being controlled, even when trained on one spacecraft configuration. Additionally, this study discusses the application specifics that yielded the reported results and discusses possible routes of expansion for future work.

INTRODUCTION

As the number of spacecraft launched and operated continues to grow, spacecraft must become increasingly autonomous. This autonomy includes the minimizing ground transmission of data and human intervention in spacecraft operations. Spacecraft are also being sent on missions where communication times to Earth can be longer than the time allowed for solving a problem or making a decision–meaning the spacecraft must perform actions based on on-board processing of data and highly capable controllers and decision-making systems.

Moreover, spacecraft are being sent to increasingly complex dynamical environments, such as NASA's Juno mission to Jupiter and OSIRIS-REx mission to asteroid Bennu.[1,2] An asteroid such as Bennu presents a weak and complex gravity field, where accurate navigation and orientation are critical for mission success. Juno presents a large multi-body gravity field, where Jupiter's many moons create a complex dynamical environment. In both cases, imperfect knowledge of the spacecraft's state must be used in concert with

[*]Graduate Student, Astrodynamics and Space Research Laboratory, Department of Aerospace Engineering and Mechanics, The University of Alabama, Tuscaloosa, AL, 35487, USA. jgelkins@crimson.ua.edu

[†]Assistant Professor, Astrodynamics and Space Research Laboratory, Department of Aerospace Engineering and Mechanics, The University of Alabama, Tuscaloosa, AL, 35487, USA. rsood@eng.ua.edu

[‡]Aerospace Engineer, Science and Technology Corporation, NASA Ames Research Center, MS 258-6, Moffett Field, CA, 94035, USA. clemens.rumpf@nasa.gov

complex navigation techniques.[3] For OSIRIS-REx, the Bennu gravity field was estimated from limited observation data pre-launch, with gravity field approximations updated using OSIRIS-REx tracking and sensing data.[4] These missions serve as exemplars of modern and future space endeavors that require an attitude control system that can adapt and even tune itself over time to localized gravity fields in an evolving dynamical environment.

As more highly capable computing power is made lighter, smaller, and more radiation-resistant, increasingly advanced software can now be launched on relatively small satellite systems.[5,6] These advancements in on-board computing hardware have made many developments in software newly feasible for on-board deployment on spacecraft, such as the recent acceleration in the field of artificial intelligence and machine learning (AI/ML). NASA has identified numerous AI research areas as enabling technologies needed for future missions in the *2020 NASA Technology Taxonomy*.[7] Examples of AI-related technologies identified in the taxonomy include intelligent robots/digital assistants, software, and on-board algorithms for data processing. As such, there is much work to be done in the practical application of modern AI research to the aerospace domain.

Reinforcement learning (RL), a paradigm of machine learning, involves an agent learning to act optimally in a changing environment by iterative training to maximize the reward received from taking actions in respective states. RL has been readily shown to achieve better-than-human performance at a variety of complex tasks and games.[8] While the development of RL applications to spacecraft control have been few and sparse, the results of preliminary studies are promising.[9–11] Gaudet et al., for instance, propose using RL as a framework for an integrated guidance and control system, and provide system performance results for a Mars descent and landing simulation, showing an increase in landing precision and robustness to noise and uncertainty.[12] RL has also been shown to produce controllers that can land and hover spacecraft in irregular or unknown gravity fields.[13,14] However, limited work has been done to study and extensively simulate spacecraft attitude control using RL. Related concurrent work includes the use of Proximal Policy Optimization (PPO) to train a spacecraft attitude controller that outperforms a quaternion rate feedback controller, and the use of a Deep Q Network and discrete control to stabilize spacecraft attitude after some momentum-changing event, such as a payload decoupling.[15,16] Other similar work includes using a neural network as a predictive attitude controller trained through generation of optimal control datasets, using a variant of dynamic programming for adaptive spacecraft attitude control.[17,18]

With both commercial and governmental interest growing in spacecraft autonomy, it is essential to explore the practical application of RL to autonomous spacecraft attitude control.[19–21] RL algorithms have typically been developed to determine optimal actions in control processes formalized as Markov decision processes (MDPs). MDPs operate in dynamics which are discrete in time, allowing states and their respective transitions to be described in detail mathematically. At each discrete timestep, the controller (agent) takes actions and receives a reward based on the value of the action, which allows the formalism of the MDP to transfer well to the formulation of RL. In this work, the task of orienting a spacecraft to a desired attitude using continuous control of torques about each of the princi-

pal body-frame axes is considered. Attitude sensor models, such as star trackers or inertial guidance units, were not used for the attitude representation. We introduce and explore the simulation techniques and RL implementation used to achieve industry-standard pointing accuracy control in performing large-angle spacecraft slew maneuvers, and demonstrate the derived controller's ability to overcome disturbance events not seen during training, such as spacecraft tumbling, impulsive disturbance torques, continuous disturbance torques, and different values of rotational inertia.

METHODS

Reinforcement Learning Formalism

RL is generally defined as optimizing a *policy* $\pi(a_t|s_t)$, a function mapping from states, s_t, to actions, a_t, by iteratively interacting with the environment to maximize reward, r_t. As discussed in the formulations of MDPs above, this requires each state and action to be discretized in time, and each action for a given state has an associated reward. The RL agent samples actions in the action space to maximize the reward returned from the environment.

Since the policy and value functions are not known prior to training, value-based, model-free RL algorithms use artificial neural networks (ANNs), often referred to as simply "neural networks," to approximate the policy and value functions. Neural networks, roughly modeled after thinking activity observed in the human brain, are multi-layered data processing systems. Each layer of a neural network contains many interconnected "neurons." Each neuron contains parameters (a *weight* and *bias*) that are collectively optimized to approximate a function, a mapping between an input to a respective output.[22] An example neural network is shown in Figure 1.

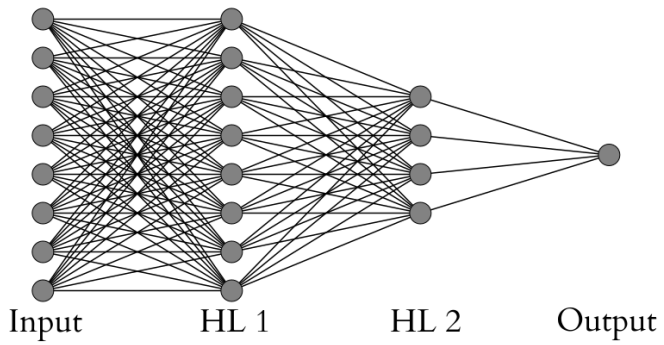

Figure 1. An example neural network with an input layer of 8 neurons, two fully-connected hidden layers (labeled HL 1 and HL 2) of 8 and 4 neurons, respectively, and a singular output.

Today's RL algorithms, based on how experience is generated by the agent, can be roughly divided into two taxa: on-policy and off-policy. On-policy methods of RL train the agent (i.e. update the networks) only using experience generated by the current network parameters, while off-policy RL utilizes any experience, often generated by previous iterations of the agent. Almost all of the algorithms in the on-policy taxon follow some

form of the actor-critic architecture, where separate networks are trained to estimate both the policy and a state-value function.[23–25] Similarly, almost all algorithms in the off-policy taxon utilize Q-learning, which is a state-action value function on the estimate of the sum of future discounted rewards if the current policy is followed to the end of some time horizon. Twin Delayed Deep Deterministic Policy Gradient (TD3), the algorithm used in this study, is a state-of-the-art off-policy, Q-based method that combines elements of the actor-critic framework by maintaining separate actor and critic networks.[26] We use this algorithm due to its sample efficiency, requiring fewer interactions with the environment to achieve desired performance, and its demonstration of performing more effectively on the OpenAI gym MuJoCo continuous control tasks than other popular methods such as PPO.

TD3 built upon known problems of the popular Deep Deterministic Policy Gradient (DDPG) algorithm.[27] DDPG combines two families of RL algorithms, Q-Learning and policy gradient, by adapting Q-Learning to a continuous action space. Q-Learning is a value-based method of learning the discounted rewards-to-go of states, while the popular Deep Q Network (DQN) algorithms are only applicable to discrete action spaces.[8,28] Policy gradient algorithms, in contrast, are formulated for RL problems with continuous action spaces, where the policy is simply optimized in the direction (gradient) of maximum reward.[29] DDPG performs this combination by learning both the policy and an action-value function.

The return from the current state, $R_t(s_t)$, at a given time, t, can be described as the discounted sum of future rewards to some time horizon, T, as

$$R_t(s_t) = \sum_{k=0}^{T} \gamma^t r_{t+k} \tag{1}$$

with the discount factor $\gamma \in (0, 1]$ weighting how much rewards from future timesteps is considered. The action-value function can then be defined as the return expected if the agent takes action a_t in state s_t and follows the current policy in each subsequent state.

$$Q^\pi = \hat{\mathbb{E}}\left[R_t | s_t, a_t\right] \tag{2}$$

Defining current network parameters as θ, deterministic policy gradient algorithms rely on taking the gradient of the expected return at $t = 0$, $J(\theta) = \hat{\mathbb{E}}\left[R_0\right]$

$$\nabla_\theta J(\theta) = \hat{\mathbb{E}}\left[\nabla_{a_t} Q^\pi(s_t, a_t) \nabla_\theta \pi_\theta(s)\right] \tag{3}$$

where the action a_t is found using policy $\pi(s_t) = a_t$. Equation 3 largely summarizes the goal of reinforcement learning–to find the policy parameters that maximize the expected returns. As mentioned above, DDPG combines Q-Learning and policy gradient algorithms into an actor-critic style algorithm, with one function approximator learning the action-value function and another learning the policy. The action-value function in Q-Learning is learned using temporal difference learning, which originates from the famous Bellman

equation of dynamic programming, where the agent learns the relationship of the value of the current state-action pair (s_t, a_t) and the state-action pair at the next timestep.[30–32]

$$Q^{\pi}(s_t, a_t) = r_t + \gamma \hat{\mathbb{E}} \left[Q^{\pi}(s_{t+1}, a_{t+1} \right] \tag{4}$$

As the action-value functions are not known beforehand, they are approximated using deep neural networks. DDPG also uses secondary "target" networks for both the policy and action-value functions, respectively denoted as $\pi_{\theta'}$ and $Q_{\psi'}(s_t, a_t)$. These target networks are stationary over some timesteps during learning, thus used as a sort of moving average to stabilize training with a fixed objective.[8]

$$y = r_t + \gamma Q_{\psi'}(s_{t+1}, a_{t+1}) \tag{5}$$

The actions in this update are selected from the target policy network $a_{t+1} = \pi_{\theta'}(s_{t+1})$. As the agent interacts with the environment, experience is stored in a replay buffer. Experience is stored in the form of "transition tuples" of (s_t, a_t, r_t, s_{t+1}), which are sampled in minibatches to update the parameters of the current policy and action-value networks. The agent interaction with the environment and replay buffer is diagrammed in Figure 2 below. Periodically during training, the network parameters of the target policy and target action-value networks are slowly updated with the current respective networks using a proportion constant:

$$\psi' \longleftarrow \tau\psi + (1 - \tau)\psi'. \tag{6}$$

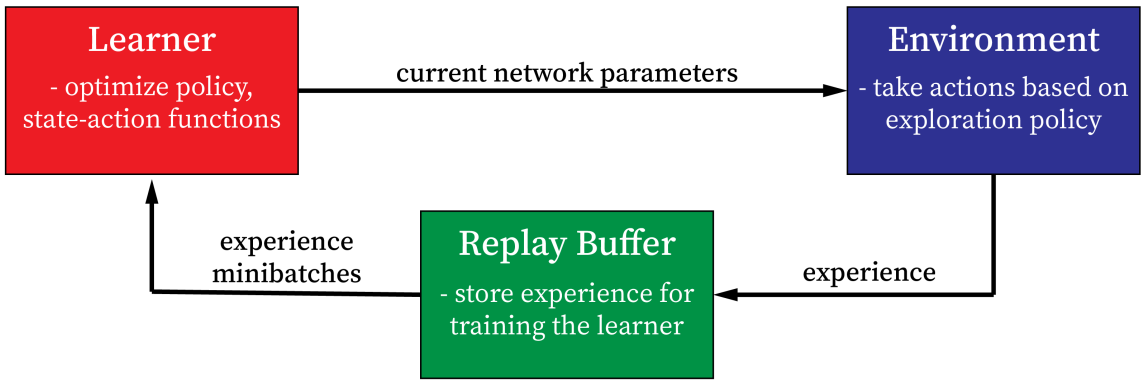

Figure 2. The agent interaction diagram for off-policy actor-critic RL algorithms such as TD3. The agent interacts with the environment to generate experience, which is sent to and stored in the replay buffer. The learner then samples experience from the replay buffer to update the networks.

TD3 corrects a major shortcoming of DDPG: overestimation bias of the state-action value, which results in the accumulation of error in this estimation of the state-action value. TD3 does this by utilizing three main differences from DDPG: using two function approximations for the state-action value, updating the policy less frequently than the value

functions (delayed policy updates), and smoothing of the target policy. By learning two different ("twin") approximations to the state-action value, TD3 reduces value overestimation by simply taking the minimum of the two predicted values at each timestep. Delayed policy updates, along with the policy parameter update equation Equation 6 above, force the policy updates to utilize a lower-variance value estimate, leading to more stable learning of the policy. Finally, during each update step, noise (clipped at some value) is added to each dimension of the target actions, which restricts the value function's tendency to exploit sharp curves in the learned value function (i.e. smoothing). Each of these improvements to DDPG put TD3 as a state-of-the-art single-agent off-policy algorithm. We employ off-policy learning for various reasons. First, off-policy learning makes use of experience replay–the storing and retraining of past interactions with the environment–to increase data efficiency.[33] Second, with most off-policy exploration being some value of noise added to actions selected from the policy, it is more easily controlled and manipulated. Third, recent advancements in distributed off-policy RL show promise, and the implementation of these advancements is planned for future extensions of this work.[34,35]

Simulating the Spacecraft Attitude Control Problem

A high-fidelity simulation environment is required for the practical application of RL. It is expensive and labor-intensive to collect enough data that accurately models the dynamics and anomalies the real-world system would experience. For complex tasks, the agent may require millions of iterations in an environment to properly explore the numerous possibilities and evolving dynamics. Given the cost of satellite launch and operation, it is thus practical to employ computer simulation, which allows for faster-than-real-time dynamics propagation for rapid training and prototyping. However, since the simulation is built as an approximation to the real-world dynamics, agents may be successful in simulation but fail to adapt to the real-world counterpart system.[36] This is an important facet in the application of RL systems to spaceflight problems, as the systems are expensive, relatively hard to replace upon failure, and the space environment is an extensive system to accurately model. The simulation environment must thus contain enough fidelity and similarity to the real-world problem to allow a seamless transition from the simulator to the actual system, such that agents' success in the simulation environment translates to success in the real-world control of the system. This workflow of training an agent in a simulation environment for transfer into the real world has been referred to as the "sim-to-real" loop in recent RL literature.[36–38] The research shows that randomized environment dynamics allow for an agent that is highly generalizable and improves agent transition from simulation to real-world operation.

The simulation environment used in this report was built using the OpenAI gym format, which allows for easy interface to the OpenAI control problems, called gym *envs*, used for rapid debugging and testing of the RL algorithms.[39] To simulate a real-world problem, a Lockheed Martin Corporation LM50 satellite bus system* was modeled, shown in Figure 3. This allows comparison of test results to publicly available pointing accuracy specifications

*https://www.lockheedmartin.com/en-us/products/satellite.html

of the satellite system, which will be used in the formulation of success criteria and reward functions in the simulation environments.

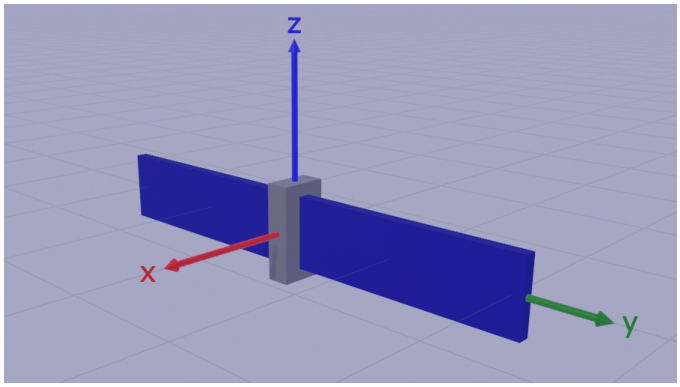

Figure 3. LM50 spacecraft model with body-fixed principal axes labeled.

We model the LM50 satellite mentioned above as symmetric about the body-fixed principal axes (all product of inertia elements equal to zero) in the rotational inertia tensor $\boldsymbol{I}$.

$$\boldsymbol{I} \equiv \text{diag}(I_1, I_2, I_3) = \text{diag}(0.872, 0.115, 0.797) \text{ kg m}^2 \tag{7}$$

The spacecraft attitude control problem is simulated by integrating Euler's rotational equations of motion for rigid body rotation

$$\vec{M} = \boldsymbol{I}\dot{\vec{\omega}} + \boldsymbol{\omega}^{\times}\boldsymbol{I}\vec{\omega} \tag{8}$$

where the skew-symmetric cross-product matrix is given by

$$\boldsymbol{\omega}^{\times} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{9}$$

Note, $\boldsymbol{I}$ is the rotational inertia tensor of the modeled spacecraft with respect to the spacecraft center of mass, $\vec{M}$ is the external moment (or *torque*) vector on the spacecraft, and $\vec{\omega}$ is the angular rotation vector about the body-fixed principal axes.[40] The subscripts 1, 2, and 3 denote the body-fixed principal axes x, y, and z, respectively.

The spacecraft attitude is represented using Euler parameters (unit quaternions). The *error quaternion* of the spacecraft (that is, the quaternion relating the current body frame of the spacecraft to the desired frame) is used to represent the spacecraft attitude to increase problem generalizability. The error quaternion $\boldsymbol{q_e}$ can be propagated through time by integrating the quaternion kinematics relation

$$\boldsymbol{q_e} = \begin{bmatrix} \hat{e}\sin(\phi/2) \\ \cos(\phi/2) \end{bmatrix} = \begin{bmatrix} \vec{q} \\ q_s \end{bmatrix} \tag{10}$$

$$\dot{\boldsymbol{q}}_{\boldsymbol{e}} = \frac{1}{2}\boldsymbol{\Omega}(\vec{\boldsymbol{\omega}})\boldsymbol{q_e} \tag{11}$$

where the augmented skew-symmetric matrix $\boldsymbol{\Omega}(\vec{\boldsymbol{\omega}})$ is written as the 4×4 block matrix

$$\boldsymbol{\Omega}(\vec{\boldsymbol{\omega}}) = \begin{bmatrix} -\boldsymbol{\omega}^{\times} & \vec{\omega} \\ -\vec{\omega}^{T} & 0 \end{bmatrix} \tag{12}$$

and ϕ is the rotation angle about the unit rotation axis vector $\hat{e}$ from the current spacecraft body frame to the desired frame, visualized below in Figure 4.[41,42]

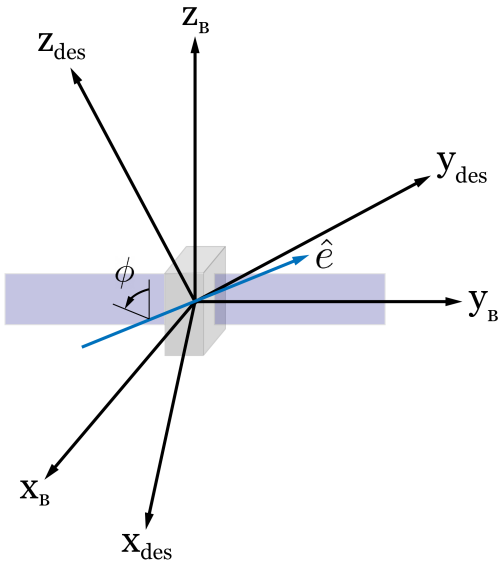

Figure 4. The relation of the spacecraft body-fixed frame, denoted as subscript *B*, to some desired reference frame, denoted as subscript *des*, about axis of rotation $\hat{e}$ and angle of rotation ϕ.

Equation (8) and Equation (11) are jointly integrated at each simulation timestep with a given torque vector $\vec{M}$ to find the spacecraft body-frame angular velocity $\vec{\omega}$ and attitude error quaternion $\boldsymbol{q_e}$ using fourth-order Runge-Kutta method.[43,44] After each integration, as computational roundoff error is introduced, the error quaternion is normalized trivially ($\boldsymbol{q_e} = \boldsymbol{q_e} / ||\boldsymbol{q_e}||$) to preserve unit magnitude.[42,44]

Problem Framework

As discussed above, the MDP (and in partially-observable state case, Partially Observable Markov Decision Process, or POMDP) is the decision-making model used to quantify RL. Central to MDPs is the discrete timestep, which requires the careful conversion of a continuous-time real-world problem to a discrete-time "episode" format. This translation requires careful examination of the desired task for the agent to complete, the state vector required to accurately represent the problem, and the reward function needed to correctly incentivize the desired agent performance.

In this work, we consider the spacecraft attitude control problem as performing a large-angle slew maneuver and subsequently maintaining a desired orientation. We define a

large-angle slew as a goal orientation that is an angle $\phi \in [30°, 150°]$ about any rotation axis $\hat{e}$ from the initial orientation. To allow for agent generalization and a closing of the sim-to-real loop mentioned above, at the start of each episode, an initial error quaternion $\boldsymbol{q_e}$ is selected from the Lie 3D rotation group *SO(3)* with the constraint, such that $\phi \in [30°, 150°]$. To generate a random orientation, a random unit vector in spherical coordinates is sampled from a uniform distribution and converted to Cartesian coordinates. The rotation angle ϕ is also sampled from a uniform distribution in the domain given above. This random unit vector (axis of rotation) $\hat{e}$ and rotation angle ϕ, now an *axis-angle* orientation representation, is then converted to the quaternion $\boldsymbol{q_e}$ using Equation (10).

In each training episode, the spacecraft is initialized at rest ($\vec{\omega}^T = [0, 0, 0]$ rad/s) in a zero-gravity environment. As mentioned above, modeling perturbation torques such as torques due to atmospheric drag, solar radiation pressure, gravity gradient forces, and forces due to the geomagnetic field (for an Earth-orbiting satellite) during training would likely shrink the sim-to-real loop. Since we are demonstrating that the agents trained can overcome perturbations unseen during training, we assume the ideal (zero-perturbation) case in this work during training.

At each timestep, the agent selects the control moment vector to enact about the spacecraft's body-fixed principal axes. Each component of this moment vector is restricted to a maximum magnitude of 0.5 Nm. This magnitude was selected to be within typical reaction wheel maximum torque bounds for the given spacecraft size. Similar to the modern Atari-playing RL implementations, we implement a variant of "frame skipping." Implementing a frame skip, in which the agent selects an action at only a set interval of frames and that action is taken until the next action frame, has been shown to greatly improve the performance of trained agents on the Atari games.[45] The variant of frame skipping used here is that the agent selects an action (torque) to rotate the spacecraft, the angular impulse is applied over the next timestep, and the system dynamics are then propagated for the following 20 timesteps with zero external torque (free rotation). This frameskip value showed a reasonable combination of desired control fidelity and training time. The integration fidelity used in this simulator is $\Delta t = 1/240$ s, similar to that of industry-standard physics engines.[46]

The state vector s_t, used to represent the problem to the agent, is critical to the success of an applied RL implementation. The state vector, which as an input to all actor and critic networks in training and acting, is given as

$$s_t = \left\{ \boldsymbol{q_e}, \dot{\boldsymbol{q}}_{\boldsymbol{e}}, \vec{\omega} \right\} \tag{13}$$

where the optimal state vector, in this case, would be where $\dot{\boldsymbol{q}}_{\boldsymbol{e}} = \vec{\omega} = \vec{0}$ and $\boldsymbol{q_e} = [0, 0, 0, 1]$. The state vector should describe the problem as simply as possible while keeping the problem relative, to simplify a real-world implementation by removing any arbitrariness (such as an inertial coordinate frame). This is why the error quaternion is used; the error quaternion is a normalized representation of current attitude to desired attitude, a representation that simplifies the problem to a single, relative coordinate frame.

To divide the simulation environment into the aforementioned discrete episode format,

episode termination criteria must be established. In this analysis, two episode termination criteria are established. The first, a timestep criterion, terminates the episode after the agent selects 500 actions, which is 10,000 timesteps with frameskipping included. This corresponds to a real-time length of 43 minutes, 45 seconds. The episode length is chosen as a value that is large enough to allow the agent reasonable time to adequately explore the environment, while small enough to keep training time relatively low. The second, a performance criterion, terminates the episode if the magnitude of angular velocity exceeds 0.5 rad/s. The performance criterion, combined with a penalty for violation (described below), allows the agent to learn to avoid violating this criterion.

The simulation environment was developed and implemented in the Python programming language, with neural networks, ADAM optimizer, and learning rate annealing implemented using the Python deep learning library, PyTorch.[47] The simulation environment integrations and other calculations were accelerated using Numba, a Python just-in-time compiling library, for reduced training time and rapid prototyping.[48] The network calculations and updates following each training epoch were done in parallel on a graphics processing unit (GPU) to further accelerate agent training. The hardware used to train the agents includes an AMD Ryzen 7 1700X eight-core CPU clocked at 3.40 GHz, with an NVIDIA GeForce RTX 2070 Super GPU.

Agent Training

In the following experiments, deep feedforward neural networks were used, though any function approximator can likely be used. The network architectures and activation functions for all networks is given in Table 3 in the Appendix, along with hyperparameters used in all experiments tabulated in Table 4. The number of layers and neurons of each layer for the actor and critic networks were chosen to be the same as originally used by the TD3 authors.[26] The appropriate selection of depth and width of neural networks is an active research area, with varying relationships shown between network depth and width.[49] The rectified linear unit (ReLU) activation function is used due to its computational efficiency and acceleration of convergence.[50,51]

The reward function, arguably the most influential part of the RL formulation, is a direct determinant of agent behavior and training time. To avoid problems associated with heavy reward engineering, we try to make the reward function as simple as the problem allows. In this specific case, the agent has evolving goals over time: first, to slew the spacecraft to within the desired error tolerance of the desired orientation; and second, to stabilize the motion about that desired orientation and remain within the desired error tolerance for an extended period of time. To reflect these evolving goals of the agent, two reward functions are utilized. The first, given below in Equation (14), is used from the beginning of the episode until the criterion $q_s \geq 0.999962$ is satisfied. This criterion corresponds to an angular error $\phi = 0.25^\circ$ about some rotation axis $\hat{e}$, the angular error tolerance tabulated for the modeled LM50 system.

$$r_t = \begin{cases} 0.1 & q_{s,t} > q_{s,t-1} \\ -0.1 & o/w \end{cases} \tag{14}$$

Here, $q_{s,t}$ is the scalar part of the error quaternion at the current timestep, and $q_{s,t-1}$ is the scalar part of the error quaternion at the previous timestep. Simplified, the agent receives a positive reward if it got closer to the goal over the previous timestep; otherwise, the agent receives a negative reward (or penalty). This reward function incentivizes the agent to decrease the angle ϕ between the current and desired orientations over consecutive timesteps, guiding the agent to maneuver towards the desired orientation. The scalar values of Equation (14) were chosen to be an order of magnitude less than Equation (15), and binary for simplicity. A possible extension to this framework may be the use of continuous values in Equation (14), such as the amount of decrease in angular error over the previous timestep.

Upon reaching the criterion $q_s \geq 0.999962$, the reward function is then switched to Equation (15) for the remainder of the episode:

$$r_t = \boldsymbol{\alpha} \cdot \boldsymbol{q_e} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} q_1^2 \\ q_2^2 \\ q_3^2 \\ q_s^2 \end{bmatrix} \tag{15}$$

where $\boldsymbol{\alpha}$ is used as a weight vector to give slightly more numerical fidelity in the error quaternion. Note that no penalties are enforced for terms of efficiency (such as the magnitude of the torque commanded), so our agent is incentivized to maneuver to the goal as quickly as possible (thus, we can expect an agent that is an approximation to time-optimality). Recall, $\boldsymbol{q_e}$ is the current attitude error quaternion. This reward function is maximized when the angle ϕ is minimized, meant to incentivize the agent stabilizing about the desired orientation. The reward of Equation (15) in the optimal state would be $r_t = 1$. This relative magnitude of 1, when compared to 0.1 from Equation (14), incentivizes the agent to decrease the attitude error to within the specified error bounds. The relative differences within Equation (15) then incentivize decreasing the attitude error as much as possible at each timestep. Additionally, a terminal reward of +10 is given if the terminal state satisifies $q_s \geq 0.999962$, 0 otherwise. A reward of -25 is given if the agent terminates the episode early (magnitude of angular velocity exceeds 0.5 rad/s).

Training Results

The initial agent (henceforth referred to as agent 1) was trained on 10,000 episodes of experience. On the hardware previously described, this 10,000-episode training run equated to a training burden of 1 day, 15 hours, 24 minutes. The training curves for the agent shown in this work are plotted in Figure 5. Agent 1 statistics are calculated and tabulated below in Table 1 for 5,000 simulated episodes, with the episode length set to 500 timesteps (43.75 seconds) for statistics calculation. The "closest state" is defined as the state when the error angle ϕ is minimum for the episode. "Terminal state" is defined as the state at the end of the simulated episode.

This agent shows desired success in the environment described above, as the agent maneuvers to and maintains an angular error $\phi < 0.25°$, which is the angular pointing accuracy reported for the LM50 system. Agent 1 shows the ability to adapt to the changing problem

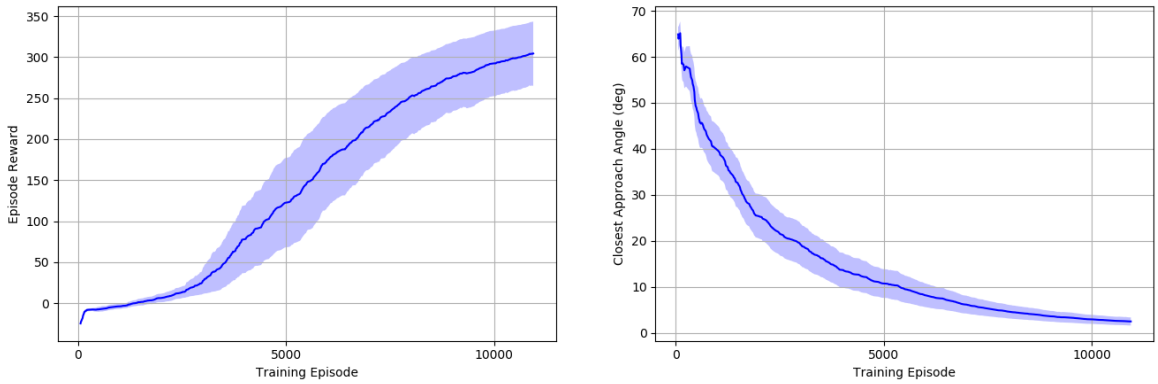

Figure 5. The training curves for this task, averaged over three seeds. The shaded region represents half a standard deviation. We calculate these results by running the agent with zero exploration noise for 10 evaluation episodes and averaging the total episode reward and minimum angle ϕ after every 10 training episodes. Curves are smoothed for visual clarity.

Table 1. Agent 1 statistics for 5,000 simulated episodes.

	Closest State		Terminal State	
	ϕ (°)	$\|\|\vec{\omega}\|\|$ (rad/s)	ϕ (°)	$\|\|\vec{\omega}\|\|$ (rad/s)
Mean	0.1301	0.0081	0.1902	0.0083
Std. dev.	0.0130	0.0004	0.0332	0.0000
Min	0.0822	0.0063	0.1254	0.0078
Q1	0.1231	0.0079	0.1571	0.0083
Q2	0.1373	0.0081	0.1593	0.0083
Q3	0.1377	0.0081	0.2235	0.0083
Max	0.1571	0.0103	0.2334	0.0084

of performing the slew maneuver and then maintaining the desired orientation, shown in the max angular error in both the closest state and terminal state being within the desired accuracy. These statistics were calculated without the presence of any perturbation torques.

To demonstrate agent control in the simulated environment, three slew maneuvers were considered with no perturbation torques included. These slew maneuvers are tabulated in Table 2. These slew maneuvers were selected to be exemplars of general agent behavior. Each slew is 100° about an axis of rotation with equal magnitude components. In each test, the rotation axis is placed in a different octant. The agent's orientation, angular error, and control histories in the simulation environment are shown in Figure 6 for the three tests considered.

Table 2. The slew maneuvers selected as control examples.

Test	Axis of Rotation	Angle (°)	Quaternion
1	[0.57735, 0.57735, 0.57735]	100	[0.44228, 0.44228, 0.44228, 0.64279]
2	[0.57735, -0.57735, 0.57735]	100	[0.44228, -0.44228, 0.44228, 0.64279]
3	[-0.57735, -0.57735, 0.57735]	100	[-0.44228, -0.44228, 0.44228, 0.64279]

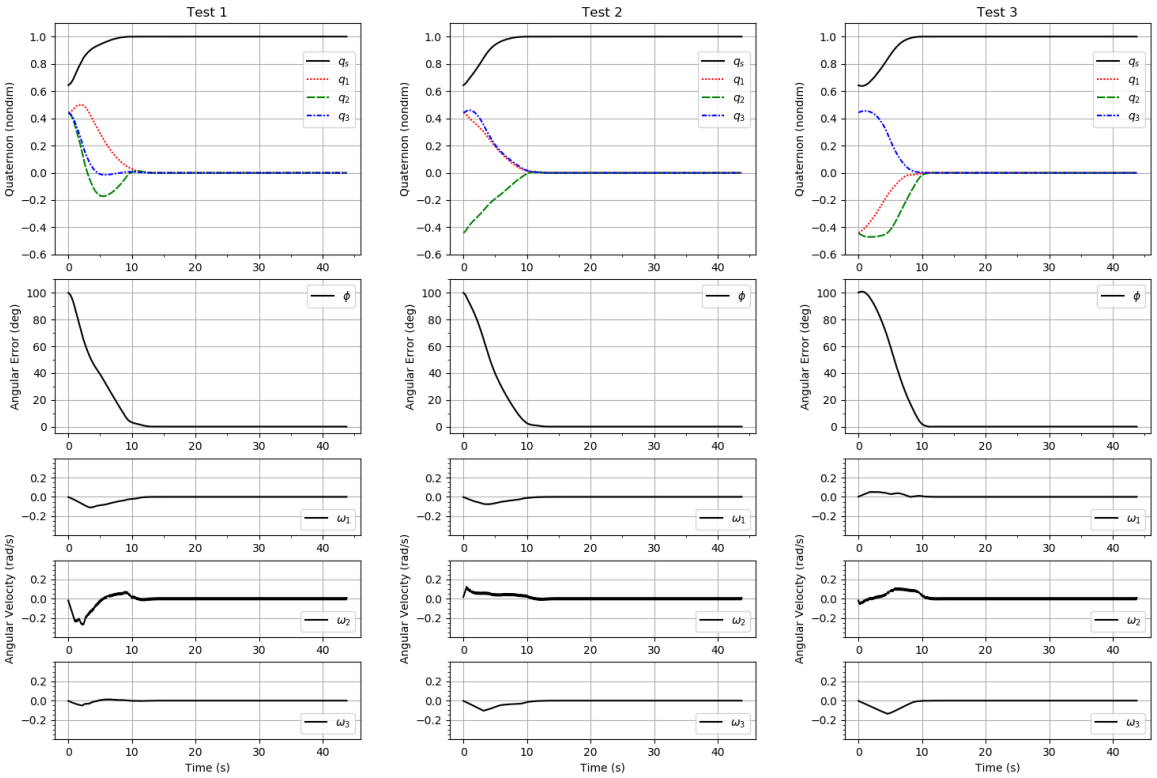

Figure 6. Control examples for the agent performing each test slew maneuver tabulated in Table 2, with no perturbation torques. The control frequency here is the same as during training, 11.43 Hz.

Resilience to Disturbance Torques

A major contribution of this work is to analyze the performance of a neural attitude controller, trained with the RL framework proposed, in the presence of various disturbance torques and initial conditions unseen during training. In this section, we analyze four different test case conditions that are of interest in evaluating our agent:

1. Spacecraft tumble (nonzero initial angular velocity, $\vec{\omega}_i \neq \vec{0}$)
2. Single, impulsive disturbance torque
3. Constant disturbance torque
4. Different rotational inertia

Each test was chosen as an example of conditions the agent may experience over the lifetime of a mission. Spacecraft detumble conditions may occur after a spacecraft awakens from an idle mode, or docks with a passive spacecraft for servicing.[52] A single disturbance torque could be the result of an impact with another object, such as another spacecraft during proximity operations, a faulty attitude control thruster that is quickly shut-off, or SmallSat deployment from a rack. A constant disturbance torque could be an environmental

disturbance such as atmospheric drag or gravity gradient forces. Finally, the inertia values of the spacecraft will change over the mission as fuel is depleted, the spacecraft is docked with another spacecraft, or a part of the spacecraft is lost due to other system failures. These are each challenges that an adaptive attitude controller must be resilient to.

In each of the tests below, the agent control frequency is increased to 40 Hz, which is implemented in our simulator by setting the frameskip number to five timesteps. This is done to better reflect the control frequency of the agent when deployed on real systems, as the training control fidelity was chosen to simply decrease training time. At each timestep, the agent selects a torque command, the torque is applied over the next timestep, and the system dynamics are then propagated for the following 5 timesteps with zero torque commanded (recall our simulation fidelity is $1/240$ s). In disturbance test 1, the initial error quaternion is set to $\boldsymbol{q_e} = [0, 0, 0, 1]$ (begins at the desired orientation). All other disturbance tests are executed with the initial error quaternion as in Test 1 of Table 2.

In the first disturbance test, the spacecraft is set in tumble-like conditions, with an initial angular velocity of $\vec{\omega}_i^T = [1.0, 2.0, 0.5]$ rad/s as in Aghili (2009).[52] Aghili presents a time-optimal detumbling of spacecraft test case, in which the spacecraft has similar maximum torque command magnitudes 0.5 Nm. We use similar conditions, only differing in the enforcement of torque constraint and rotational inertia tensor values. Aghili restricts the magnitude of the commanded torque vector to be less than or equal to 0.5 Nm, where we restrict each component of the torque vector to be less than or equal to 0.5 Nm in magnitude. We also keep the rotational inertia matrix the same as during agent training to isolate the effect of the tumbling conditions. The orientation and angular velocities over time of the test are shown in Figure 7, labeled "Disturbance Test 1." Our test results show that the agent achieves 99.87% passivation at $t = 19.34$ s, which is the time-optimal complete passivation time in Aghili (2009). This suggests our agent to be a near-approximation of time-optimal attitude control, especially interesting in initial conditions not explicitly seen during training.

In the second disturbance test, initial conditions are again equivalent to Test 1 in Table 2, with the agent performing the 100° slew maneuver. In this test, a singular disturbance torque of the value $\vec{\tau}_{dist}^T = [5.0, 2.0, 1.0]$ Nm is enacted on the spacecraft at $t = 15$ s into the episode. The value of this torque was chosen to be large enough to provide challenging conditions and interesting visual results, with disturbance torque magnitudes differing about each principal axis. We find that the agent is able to successfully recover from the disturbance and return to the desired goal of $\phi \leq 0.25^\circ$. The orientation and angular velocities over time for this disturbance test are also shown in Figure 7, labeled "Disturbance Test 2."

In the third disturbance test, we wanted to mimic a challenging constant disturbance torque that was reflective of space environment conditions. Due to the complications of drag being a function of ram area, we opted to implement a disturbance similar to that of gravity gradient. For the spacecraft modeled, low-Earth orbit gravity gradient disturbances are almost negligible, so the disturbance magnitude was increased for our tests for interest. The order of magnitude of the disturbance was selected to be consistent with the gravity-

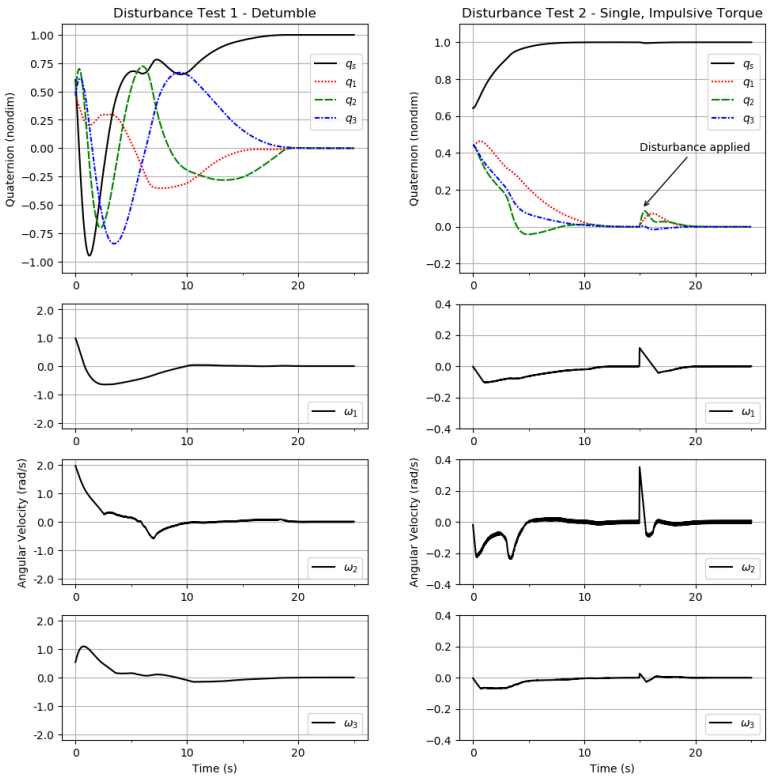

Figure 7. Control examples for the agent performing the first two disturbance test cases. The agent control frequency here is 40 Hz.

gradient forces experienced by Juno near perijove, an approximate upper bound of gravity-gradient type disturbances.[53] We retain the difference in rotational inertia terms to maintain the disturbance torque's dependence on the spacecraft inertia as in gravity gradient. The disturbance torque we enact at each timestep is thus

$$\vec{\tau}_{dist} = 0.05 \begin{bmatrix} I_3 - I_2 \\ I_1 - I_3 \\ I_2 - I_1 \end{bmatrix} = \begin{bmatrix} 0.0341 \\ 0.0038 \\ -0.0379 \end{bmatrix} \text{Nm} \tag{16}$$

where the disturbance is applied in the same inertial frame direction at each timestep, similar to that of gravity gradient torque. We again find that the agent is able to achieve desirable performance under the influence of this disturbance unseen during training, without the need for updating or retraining the policy. The result of this test is particularly interesting, as we implement the frameskipping described above, the state-transition tuple the agent "expects" is different than in training, as the disturbance torque is able to act in the timesteps when the agent does not. The policy being robust to this disturbance is intriguing. The orientation and angular velocities over time of this test are shown in Figure 8, labeled "Disturbance Test 3."

In the fourth and final disturbance test we conducted, the rotational inertia tensor is de-

creased elementwise by a factor of two. This was done to study robustness to the system's mass changing over the course of a mission, as a resource such as fuel is depleted. In this test, we found that the agent is easily able to maintain control of the spacecraft with the modified inertia tensor. We also tried tests in changing the inertia tensor values themselves to change the symmetry of the spacecraft, with similar results. This suggests that the learned policy is not only agnostic to the magnitude of this system's rotational inertia tensor, but spacecraft agnostic–the controller can feasibly perform on systems it was not trained on. The orientation and angular velocities over time of this test are shown below in Figure 8, labeled "Disturbance Test 4."

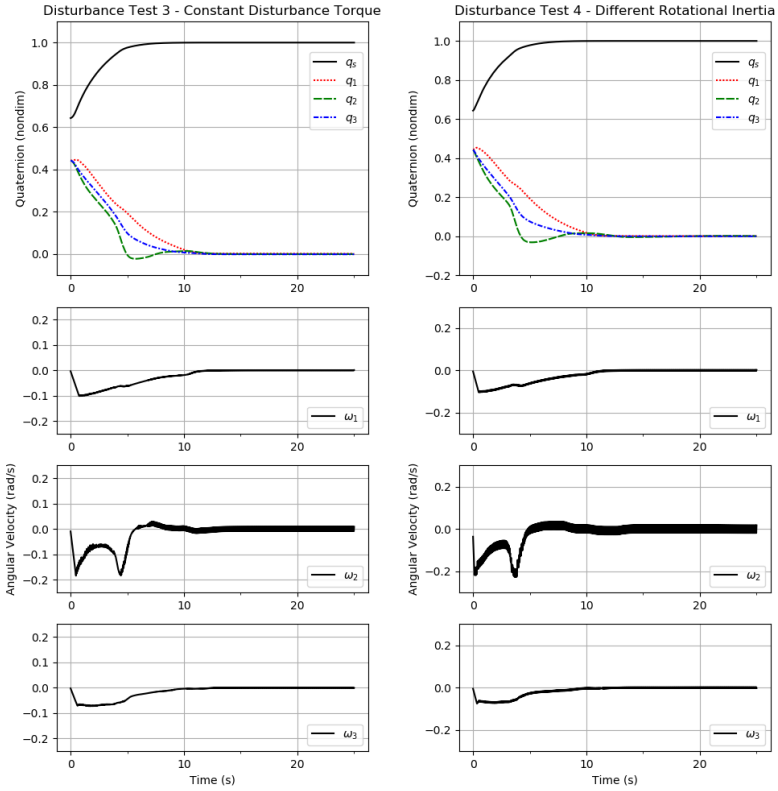

Figure 8. Control examples for the agent performing the second two disturbance test cases. The agent control frequency here is again 40 Hz.

It is important to note that the "final product" of training is the learned policy $\pi(a_t|s_t)$ which, given our state and action representation, is a neural network generally mapping the quaternion, quaternion rates, and angular rates to body-fixed axis torque commands. The ability for the derived controller to be robust to conditions unseen during training is important–a large issue of utilizing simulation to perform machine learning is that it is often infeasible to account for every possible circumstance the system will encounter in operation. Thus, framing the state vector in such a way that produces a generalizable policy is important. These tests help to prove the feasibility of neural attitude controllers in real-world operation. Secondly, the learned policy being system-agnostic could suggest the use of a "master" policy, one single well-trained controller for all satellites in a constellation.

CONCLUSIONS AND FUTURE WORK

This work presents a feasible framework for training a neural spacecraft attitude controller using reinforcement learning, investigates training techniques and reward functions that can be used to improve control accuracy, and demonstrates that these controllers can be trained on personal-computer strength hardware. A state-of-the-art single-actor reinforcement learning algorithm is implemented and applied on the developed simulation environment, where the final trained agent is shown to achieve industry-standard pointing accuracy in a relatively short amount of training time. We then test the agent's resilience to environmental disturbances in four different test cases, each designed to simulate a different condition the agent could encounter in the space environment. The agent is able to successfully adapt to all disturbance tests conducted, even showing a close approximation to time-optimality. The ability for the agent to be robust to conditions not explicitly seen during training helps prove the feasibility of using RL-powered controllers on real satellite systems. The results also suggest that it could be possible to use a single "master" policy across a wide range of satellites, which could help increase spacecraft constellation autonomy, a necessary step for the space exploration endeavors of the future. Expansion of this work aims to utilize recent advancements in distributed reinforcement learning to study how a satellite constellation generating data may be used for tasks such as spacecraft attitude control.

REFERENCES

[1] S. Bolton, J. Lunine, D. Stevenson, J. Connerney, S. Levin, T. Owen, F. Bagenal, D. Gautier, A. Ingersoll, G. Orton, *et al.*, "The Juno mission," *Space Science Reviews*, Vol. 213, No. 1-4, 2017, pp. 5–37.

[2] D. Lauretta, S. Balram-Knutson, E. Beshore, W. V. Boynton, C. D. d'Aubigny, D. DellaGiustina, H. Enos, D. Golish, C. Hergenrother, E. Howell, *et al.*, "OSIRIS-REx: sample return from asteroid (101955) Bennu," *Space Science Reviews*, Vol. 212, No. 1-2, 2017, pp. 925–984.

[3] B. Williams, P. Antreasian, E. Carranza, C. Jackman, J. Leonard, D. Nelson, B. Page, D. Stanbridge, D. Wibben, K. Williams, *et al.*, "OSIRIS-REx flight dynamics and navigation design," *Space Science Reviews*, Vol. 214, No. 4, 2018, p. 69.

[4] D. Scheeres, A. French, J. McMahon, A. Davis, D. Brack, J. Leonard, P. Antreasian, M. Brozovic, S. Chesley, D. Farnocchia, *et al.*, "OSIRIS-REx Gravity Field Estimates for Bennu using Spacecraft and Natural Particle Tracking Data," 2019.

[5] F. Bruhn, K. Brunberg, J. Hines, L. Asplund, and M. Norgren, "Introducing radiation tolerant heterogeneous computers for small satellites," *2015 IEEE Aerospace Conference*, IEEE, 2015, pp. 1–10.

[6] D. Lüdtke, K. Westerdorff, K. Stohlmann, A. Börner, O. Maibaum, T. Peng, B. Weps, G. Fey, and A. Gerndt, "OBC-NG: Towards a reconfigurable on-board computing architecture for spacecraft," *2014 IEEE Aerospace Conference*, IEEE, 2014, pp. 1–13.

[7] D. Miranda, "2020 NASA Technology Taxonomy," 2020.

[8] 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, pp. 529–533.

[9] D. Izzo, M. Märtens, and B. Pan, "A survey on artificial intelligence trends in spacecraft guidance dynamics and control," *Astrodynamics*, 2018, pp. 1–13.

[10] A. Harris and H. Schaub, "Towards Reinforcement Learning Techniques for Spacecraft Autonomy," *42nd Annual AAS Guidance, Navigation and Control Conference*, 2018, pp. 18–078.

[11] A. Harris, T. Teil, and H. Schaub, "Spacecraft Decision-Making Autonomy Using Deep Reinforcement Learning," *29th AAS/AIAA Space Flight Mechanics Meeting, Hawaii*, 2019, pp. 1–19.

[12] B. Gaudet, R. Linares, and R. Furfaro, "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Landing," *Advances in Space Research*, 2020.

[13] B. Gaudet, R. Linares, and R. Furfaro, "Adaptive Guidance and Integrated Navigation with Reinforcement Meta-Learning," *Acta Astronautica*, 2020.

[14] B. Gaudet, R. Linares, and R. Furfaro, "Six Degree-of-Freedom Hovering over an Asteroid with Unknown Environmental Dynamics via Reinforcement Learning," *AIAA Scitech 2020 Forum*, 2020, p. 0953.

[15] J. T. Allison, M. West, A. Ghosh, *et al.*, "Reinforcement learning for spacecraft attitude control," *Proceedings of the International Astronautical Congress, IAC*, Vol. 2019, International Astronautical Federation, IAF, 2019, pp. IAC–19_C1_IP_4_x49857.

[16] Y. Wang, Z. Ma, Y. Yang, Z. Wang, and L. Tang, "A New Spacecraft Attitude Stabilization Mechanism Using Deep Reinforcement Learning Method," 2019.

[17] J. D. Biggs and H. Fournier, "Neural-network-based optimal attitude control using four impulsive thrusters," *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 2, 2020, pp. 299–309.

[18] Y. Zhou, E. v. Kampen, and Q. P. Chu, "Adaptive spacecraft attitude control with incremental approximate dynamic programming," *Proceedings of the IAC 2017*, 2017.

[19] N. NASA, "Technology Roadmaps: TA 5: Communications," *Navigation and Orbital Debris Tracking and Characterization Systems*.

[20] T. Fong, "Autonomous Systems: NASA Capability Overview," 2018.

[21] C. L. Moore, L. Underwood, F. Figueroa, M. G. Walker, and J. Morris, "NASA Platform for Autonomous Systems (NPAS)," 2019.

[22] M. H. Hassoun *et al.*, *Fundamentals of artificial neural networks*. MIT press, 1995.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *International conference on machine learning*, 2016, pp. 1928–1937.

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," *International conference on machine learning*, 2015, pp. 1889–1897.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[26] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[28] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, Vol. 8, No. 3-4, 1992, pp. 279–292.

[29] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.

[30] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*. Princeton university press, 2015.

[31] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, Vol. 3, No. 1, 1988, pp. 9–44.

[32] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[33] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, Vol. 8, No. 3-4, 1992, pp. 293–321.

[34] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[35] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," *International conference on learning representations*, 2018.

[36] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," *CoRR*, Vol. abs/1710.06537, 2017.

[37] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 8973–8979.

[38] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[40] F. L. Markley and J. L. Crassidis, *Fundamentals of spacecraft attitude determination and control*, Vol. 33. Springer.

[41] A. Bani Younes and D. Mortari, "Derivation of All Attitude Error Governing Equations for Attitude Filtering and Control," *Sensors*, Vol. 19, No. 21, 2019, p. 4682.

[42] F. L. Markley, "Attitude error representations for Kalman filtering," *Journal of guidance, control, and dynamics*, Vol. 26, No. 2, 2003, pp. 311–317.

[43] M. S. Andrle and J. L. Crassidis, "Geometric integration of quaternions," *Journal of Guidance, Control, and Dynamics*, Vol. 36, No. 6, 2013, pp. 1762–1767.

[44] S. R. Buss, "Accurate and efficient simulation of rigid-body rotations," *Journal of Computational Physics*, Vol. 164, No. 2, 2000, pp. 377–406.

[45] A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen, "Frame skip is a powerful parameter for learning to play atari," *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[46] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," *GitHub repository*, 2016.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[48] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6.

[49] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," *Conference on learning theory*, 2016, pp. 907–940.

[50] K. Hara, D. Saito, and H. Shouno, "Analysis of function of rectified linear unit used in deep learning," *2015 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015, pp. 1–8.

[51] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, Vol. 3, 2014.

[52] F. Aghili, "Time-optimal detumbling control of spacecraft," *Journal of guidance, control, and dynamics*, Vol. 32, No. 5, 2009, pp. 1671–1675.

[53] W. Folkner, L. Iess, J. Anderson, S. Asmar, D. Buccino, D. Durante, M. Feldman, L. G. Casajus, M. Gregnanin, A. Milani, *et al.*, "Jupiter gravity field estimated from the first two Juno orbits," *Geophysical Research Letters*, Vol. 44, No. 10, 2017, pp. 4694–4700.

APPENDIX A: HYPERPARAMETERS AND NETWORK ARCHITECTURES

Table 3. Architectures and activation functions for the actor and critic networks for this experiment, used as the same in the original TD3 paper.[26]

	Actor Networks		Critic Networks	
	# of Neurons	Activation Function	# of Neurons	Activation Function
Hidden Layer 1	400	ReLU	400	ReLU
Hidden Layer 2	300	ReLU	300	ReLU
Output Layer	3 (# actions)	Tanh	1	Linear

Table 4. TD3 hyperparameters used for training the agent. Arrow indicates linear annealing over the training run.

Hyperparameter	Value
Discount (γ)	0.99
Batch size	100
Exploration noise	0.1
Exploration noise clip	0.5
Policy delay	2
Polyak averaging proportion	0.995
Learning rate	$3 \times 10^{-4} \rightarrow 1 \times 10^{-6}$