71st International Astronautical Congress (IAC) – The CyberSpace Edition, 12-14 October 2020.

IAC–20–C1.9.8

Autonomous Spacecraft Attitude Control Using Deep Reinforcement Learning

Jacob G. Elkins[a]*, Rohan Sood[b], Clemens Rumpf[c]

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

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

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

* Corresponding Author

Abstract

While machine learning and spacecraft autonomy continue to gain research interest, significant work remains to be done in efficiently applying modern machine learning techniques to problems in spaceflight. This study presents a framework for deriving a discrete neural spacecraft attitude controller using reinforcement learning, a paradigm of machine learning, without the need for high-performance computing. The developed attitude controller is an approximately time-optimal solution to a highly constrained control problem, able to achieve well above industry-standard pointing accuracies. Control examples are also presented of the agent performing large-angle spacecraft slews in the developed simulation environment and future extensions of this work are discussed.

Keywords: Reinforcement learning, artificial intelligence, attitude control, satellite, intelligent systems

1. Introduction

High-performance spacecraft autonomy is essential to the future of space exploration [1, 2]. As humanity sends an increasing number of spacecraft on deep-space missions, with long communication times and limited bandwidth, the ability for the spacecraft to process data on-board and to make autonomous decisions becomes near critical. Modern artifical intelligence (AI) and machine learning (ML) is currently a highly-active research area that has proven to be a promising avenue for achieving advanced autonomy. An important component of autonomous decision making and control is the ability to understand long-term temporal dependencies in an environment, and recent AI/ML research results show systems that exhibit long-term planning and awareness [3]. While spacecraft attitude control has been performed autonomously for decades, many methods of traditional feedback control do not exhibit any long-term temporal understanding of the control problem. The relative simplicity of the spacecraft attitude control problem make it well suited to be used as a basis for studying how recent progress in AI/ML research might be harnessed to produce systems that may outperform traditional methods in situations where long-term planning and situational awareness is required.

Reinforcement learning (RL) is a general framework of modeling behavior by learning from a reward signal, received from interacting with the environment. In machine learning, this involves an agent taking actions in respective states to maximize future expected reward. RL is one of today's primary AI/ML research areas, gaining large amounts of research interest by outperforming humans in various games, especially those that require long-term strategy [4–6].

Another desirable quality for spacecraft autonomy is the ability for the system to perform online learning, improving its performance over time. An example of the desirability of online learning would be a spacecraft tuning its attitude controller in-situ for efficiency after capturing a large asteroid. Although current RL algorithms require considerable amounts of environment exploration and are highly resource intensive, recent research results in distributed reinforcement learning show a significant decrease in time required to train extremely effective agents [7–9]. With spacecraft being launched in groups and an increasing number of constellations, distributed RL may be able to utilize the multitude of data and experience generated by spacecraft, such as performing variants of online learning. However, implement-

ing distributed RL requires a clear, working framework of the problem being solved. Formulating a real-world problem into the format modern RL algorithms require is nontrivial, as most RL advancements are benchmarked on tasks with well-developed state representations and reward functions [10–12]. RL frameworks require careful consideration of how to represent the problem to the agent, and how to reward the agent to incentivize the desired behavior. Building upon the framework and agent presented in [13], this study helps relieve the boundary on future distributed RL work by presenting an efficient single-agent RL framework for spacecraft attitude control, a task that all spacecraft must perform effectively.

The results of related RL applications to spacecraft control and decision making have been encouraging [14–18]. Closely related concurrent work includes the use of Proximal Policy Optimization (PPO), the algorithm used in this work, for continuous spacecraft attitude control, that is shown to outperform a quaternion rate feedback controller [19]. Other work includes Deep Q Networks for discrete control [20], predictive attitude control using optimal control datasets [21], and approximate dynamic programming [22]. This work builds upon [13], which used Twin Delayed Deep Deterministic Policy Gradient (TD3) to train a neural attitude controller that can successfully adapt to perturbations unseen during training. In [13], a relatively simple reward structure was used with the highly capable RL algorithm, TD3, in continuous control. The agent trained in [13] was shown to be robust to external disturbances that were not seen during training. However, converting the attitude control problem to low-dimension discrete control results in a highly constrained control problem, and the simple rewards in [13] used with PPO do not yield acceptable results. PPO is a beneficial algorithm due to its built-in learning stability and relative computational simplicity [23]. In this work, we expand and modify the reward structure from [13] to achieve desirable results with PPO in the discrete control scenario.

To utilize the state-of-the-art advancements in distributed RL, the simulation, state vector, reward functions, and single-agent RL algorithms must be developed, which is the purpose of this study. The task of orienting a spacecraft to a desired attitude using only discrete torques about each of the principal body-frame axes is considered. In this analysis, attitude sensor models, such as star trackers or inertial guidance units, were not used for the attitude representation. This work will introduce and explore the simulation techniques, state representations, reward functions, policy and value network architectures, and algorithm hyperparameters used to achieve industry-standard pointing accuracy control in large-angle spacecraft slew maneuvers using highly constrained discrete control.

2. Methods

2.1. Reinforcement Learning Formalism

RL is a generalized learning framework, whose origination comes from the biomimicry of how humans and animals learn by interacting with the environment, simply *learning from interaction*. With the onset of high-performnace computing, computational reinforcement learning is now seen as a promising method for machine learning and artificial intelligence. Computational reinforcement learning is, at its core, learning how to map situations (states) to actions [24]. This state-action process is often formulated as a Markov Decision Process (MDP). MDPs are discrete-time processes that are defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ is the environment's state space, $\mathcal{A}$ is the agent's action space, $p(s_{t+1}|s_t, a_t)$ is the state-action transition function, and $r(s_t, a_t)$ is the state-action reward function. At each discrete timestep t, the agent selects an action from the policy $\pi(a_t|s_t)$ that maps the state space $\mathcal{S}$ to the action space $\mathcal{A}$. This action transitions the environment to the next state s_{t+1} and yields a reward r_t according to functions p and r, respectively. The MDP models the RL implementations of today very well. The RL agent samples actions in the action space to maximize the reward returned from the environment.

Many of today's most successful RL algorithms, such as the popular actor-critic family of algorithms, make use of some value function, here the state-value function $V(s_t)$ [23, 25, 26]. The state-value function maps the current state to the *value* of the current state. A commonly used state-value function is the expected value of return, R_t, shown in Equation (1).

$$V(s_t) = \hat{\mathbb{E}}\left[R_t(s_t)\right] \tag{1}$$

Return, or the sum of discounted future rewards, can be written as

$$R_t(s_t) = \sum_{k=0}^{T} \gamma^t r_{t+k} \tag{2}$$

where return at state s_t is denoted as $R_t(s_t)$, and the time horizon denoted as T. The discount factor $\gamma \in (0, 1]$ is a hyperparameter that is tuned during training. The discount factor numerically controls

the amount of future rewards taken into account by the agent. The state-value function of Equation (1), used in [25], is also employed in the following experiments.

As discussed above, computational "learning" is learning the policy and value functions. Since these functions are not known, they must be approximated. The use of deep neural networks as heavy-duty nonlinear function approximators has had a significant impact on reinforcement learning. RL is frequently referred to as "deep reinforcement learning" when deep neural networks are used, as in this study. Neural networks are bio-inspired data processing systems consisting of multiple layers of interconnected neurons, which are iteratively optimized to approximate the mapping from data inputs to outputs [27]. An example neural network diagram is shown in Figure 1.

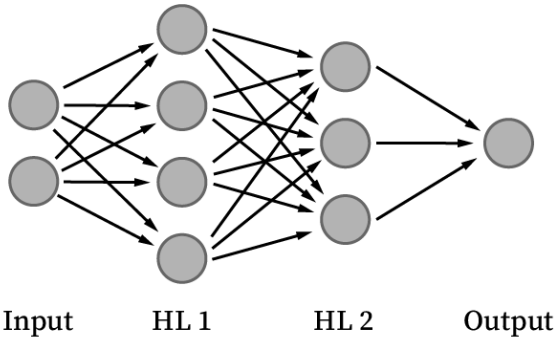

Fig. 1. An example fully connected feed-forward neural network. The grey circles represent neurons, the arrows represent the flow of data points. The input would be a vector, such as a state vector. The two hidden layers, labeled HL 1 and HL 2, are proportional to the neural network architectures used in this study.

In the following experiments, a variant of Proximal Policy Optimization (PPO) is used [23]. To represent the policy and value functions, fully-connected feed-forward ANNs (often called multilayer perceptrons, or MLPs) are used. In the PPO-clip family of algorithms, a "clip" is performed to the loss function to limit the magnitude of policy updates, which improves stability of convergence during training [23]. PPO is based on the Trust Region Policy Optimization (TRPO) algorithms, which make use of a loss function of a policy ratio and advantage product, shown in Eq. (3):

$$L(\theta) = \hat{\mathbb{E}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \tag{3}$$

where θ denotes the new policy network parameters following and update from training, $\hat{A}_t$ is some advantage estimator, and θ_{old} denotes the policy network parameters prior to the training update [28]. However, when loss $L(\theta)$ is large, the policy ratio is large; that is, the policy network receives a large update, which could lead to instability in the convergence to an optimal policy. PPO edits this loss function to including a "clipping parameter" ϵ, shown in Equation (4):

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}\Big[\min\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t, \text{clip}\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\Big)\hat{A}_t\Big)\Big] \tag{4}$$

where $1 \pm \epsilon$ can be thought of as a maximum magnitude policy update. The clipping parameter ϵ typically takes values in the range of $(0.1, 0.3)$ to help limit large, destructive policy updates. In this implementation of PPO, Eq. (4) is used as the loss function for the policy network. For this policy network loss function, an entropy bonus term $cS[\pi_\theta](s_t)$ is added, where c is a hyperparameter $c \in [0.0001, 0.001]$, which ensures sufficient exploration of the action space [23]. Typical mean squared error (MSE) is used as the loss function for the value network. Separate loss functions for the value and policy functions are used due to the networks not sharing any parameters. If parameters were shared, the loss functions would have to be combined to update the parameters appropriately. An advantage estimator $\hat{A}_t$ is used as the difference between the value return $R_t(s_t)$ of Eq. (2) and the current value network prediction $V_t^\theta(s_t)$ shown in Eq. (5).

$$\hat{A}_t = R_t(s_t) - V_t^\theta(s_t) \tag{5}$$

The agent then interacts with the environment (here, the simulated environment) for a number of episodes M to return a minibatch dataset of states, actions, rewards, and associated values. The workflow of this on-policy minibatch building is shown in Figure 2. This minibatch is then passed to a training loop where batches are sampled for N epochs to update the value and policy networks using backpropagation and an optimizer such as RMSProp or Adam [29, 30]. The Adam optimizer is used in this study. The formulation of algorithms such as PPO rely on this segmented training iteration in the environment, commonly referred to as *episodes*. Converting real-world problems and tasks, such as spacecraft attitude control, into these starting-and-ending episodes

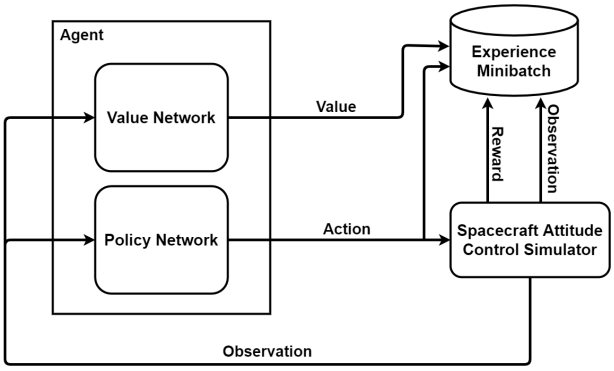

Fig. 2. Diagram of agent-environment interaction used to build minibatches of experience used in agent training.

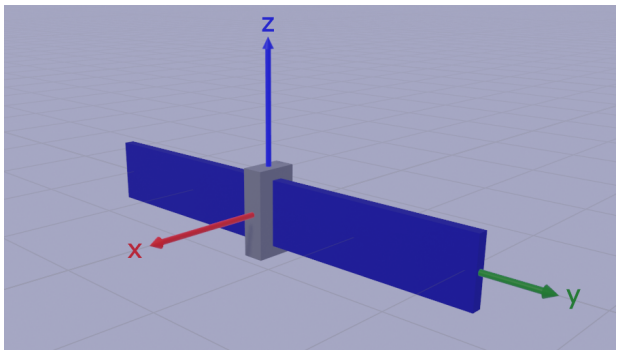

Fig. 3. Simulation rendering of the LM50 spacecraft model, with body-fixed principal axes labeled [13].

requires care and thought. As detailed above, the success of the RL implementation and its associated training time is dependent upon, among others: state representation, reward function, policy and value representation, episode termination, and action space. The exploration and proper representation of these dependencies for training a spacecraft attitude controller is the main purpose of the following experiments and analysis.

2.2. Simulation of Spacecraft Attitude

Reinforcement learning is resource intensive, requiring many interactions with the environment to generate data for the agent. Because generating real-world data can be costly and impractical, simulations are used to train an agent before possible deployment onto a real-world system. For the agent to perform adequately on a real-world system, the simulation environment should be randomized and adequately represent the real-world [31, 32]. We use the same rotational inertia tensor as [13], which is a model of the Lockheed Martin Corporation LM50 satellite bus. The LM50 bus is depicted in Figure 3. The tabulated pointing accuracies for the LM50 system were used to validate our controller's performance and feasibility relative to industry standards.

By modeling the spacecraft as a rigid body, spacecraft attitude control and rotation can be simulated by integrating Euler's rotational equations of motion

$$\vec{M} = \boldsymbol{I}\dot{\vec{\omega}} + \boldsymbol{\omega}^{\times}\boldsymbol{I}\vec{\omega} \tag{6}$$

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

$$\boldsymbol{\omega}^{\times} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{7}$$

where $\boldsymbol{I}$ is the rotational inertia tensor of the spacecraft with respect to the center of mass (in units of kg-m^2), $\vec{M}$ is the external moment (or *torque*) vector on the spacecraft directly controlled by the agent, and $\vec{\omega}$ is the angular rotation vector about the body-fixed principal axes [33]. The subscripts 1, 2, and 3 denote the body-fixed principal axes x, y, and z, respectively.

The spacecraft attitude is using unit quaternions, or Euler parameters. Specifically, we use the error quaternion, which relates the current body-fixed reference frame of the spacecraft to the desired reference frame. The error quaternion $\boldsymbol{q_e}$ is propagated through time by integrating the quaternion kinematics relation

$$\boldsymbol{q_e} = \begin{bmatrix} \hat{e}\sin(\phi/2) \\ \cos(\phi/2) \end{bmatrix} = \begin{bmatrix} \vec{q} \\ q_s \end{bmatrix} \tag{8}$$

$$\dot{\boldsymbol{q}}_{\boldsymbol{e}} = \frac{1}{2}\boldsymbol{\Omega}(\vec{\omega})\boldsymbol{q_e} \tag{9}$$

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

$$\boldsymbol{\Omega}(\vec{\omega}) = \begin{bmatrix} -\boldsymbol{\omega}^{\times} & \vec{\omega} \\ -\vec{\omega}^{T} & 0 \end{bmatrix} \tag{10}$$

and ϕ is the rotation angle about the unit rotation axis vector $\hat{e}$. The relationship between the current spacecraft body frame and the desired frame is visualized below in Figure 4 [34, 35].

Eq. (6) and Eq. (9) are integrated at each timestep with a given moment vector $\vec{M}$ to find the spacecraft body-frame angular velocity $\vec{\omega}$ and attitude error quaternion $\boldsymbol{q_e}$ at the next timestep. All numerical

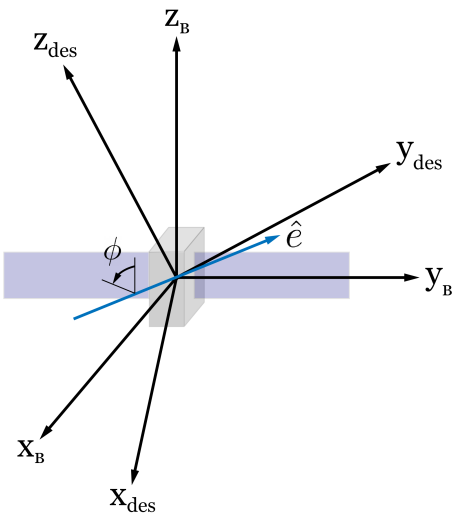

Fig. 4. The relation of the spacecraft body-fixed frame, denoted as subscript B, to some desired reference frame, denoted as subscript *des*, about axis of rotation $\hat{e}$ and angle of rotation ϕ [13].

integrations are done using fourth-order Runge-Kutta method [36, 37]. After each integration, the error quaternion is normalized trivially ($\boldsymbol{q_e} = \boldsymbol{q_e}/\,||\boldsymbol{q_e}||$) to preserve unit magnitude [35, 37].

2.3. RL Problem Setup

The MDP formulation requires RL simulation environments to be broken into discrete intervals, called episodes. Formulating a real-world problem into episode format requires careful consideration of the task at hand and the desired agent performance. The goal is to produce an attitude controller (agent) that can perform large-angle slews and stabilize the spacecraft about the desired orientation within industry-standard pointing accuracy. In this work, we define a large-angle slew to be a goal orientation that is an angular error of $\phi \in [30°, 150°]$ from the initial orientation.

The state vector numerically represents the problem to the agent, acting as the input to the policy and value functions. The state vector, one of the main determinants of agent performance, should fully represent the problem or process in which the agent is acting. The state vector s_t in this implementation is given as

$$s_t = \left\{\boldsymbol{q_e}, \dot{\boldsymbol{q}}_{\boldsymbol{e}}, \vec{\omega}\right\} \tag{11}$$

where the goal state vector is when the spacecraft is stabilized at rest about the desired orientation, $\boldsymbol{q_e} = [0, 0, 0, 1]$ and $\dot{\boldsymbol{q}}_{\boldsymbol{e}} = \vec{\omega} = \vec{0}$. We observed two small implementation details that improve agent training. First, the components of the state vector should be normalized to the order of 1. This is due to the initialization of weights in today's Python neural network libraries, such as PyTorch or Tensorflow. When the policy or value functions, represented here as neural networks, receive inputs differing in orders of magnitude, the neural networks can often map to maximum or minimum output values, crippling agent learning. Second, the state vector should be built using relative representations of the state whenever possible. Examples include using the difference in position or the error quaternion, in contrast to absolute position or some arbitrary attitude reference frame. These relative values seem to make the function approximation (i.e. the learning) much more successful.

At episode initialization, the initial error quaternion $\boldsymbol{q_e}$ is selected from the Lie 3D rotation group *SO(3)*. To ensure a uniform distribution of sampled orientations, we sample a random unit vector for the axis of rotation in spherical coordinates from a uniform distribution and and convert the vector to Cartesian coordinates. The rotation angle $\phi \in [30°, 150°]$ is also sampled from a uniform distribution. This axis of rotation $\hat{e}$ and rotation angle ϕ, now in *axis-angle* representation, are then converted to our initial error quaternion using Eq. (8). The spacecraft is initialized at rest ($\dot{\boldsymbol{q}}_{\boldsymbol{e}} = \vec{\omega} = \vec{0}$), with no perturbation or gravitational torques.

The control problem is discretized by only allowing the agent to control near-impulsive torques on any of the three principle axes in the spacecraft body frame to three magnitudes: 0.5 Nm, 0.05 Nm, or 0.005 Nm. These magnitudes were selected to be within typical reaction wheel maximum torque bounds for the size of the modeled spacecraft [38]. The lower magnitude torque values give the agent additional control authority for stabilizing about the desired attitude. The agent, in our results, frequently selects torques from all three of the above magnitudes in a given episode. A summary of possible agent actions (moment vectors) is given in Table 1. Note that the agent can only choose one body-fixed axis about which to enact external moment at each timestep, resulting in a highly constrained control problem.

Additionally, a variant of "frameskipping" is implemented in this study, as in [13]. RL algorithms and advancements are commonly benchmarked and compared by playing Atari games, the popular arcade games [25, 39, 40]. Atari games run at a frequency of 60 frames per second, where the agent could submit

Table 1. Possible agent actions at each timestep.

Action	$\vec{M}^T$ (Nm)
0	[0, 0, 0]
1, 2	[±0.5, 0, 0]
3, 4	[0, ±0.5, 0]
5, 6	[0, 0, ±0.5]
7, 8	[±0.05, 0, 0]
9, 10	[0, ±0.05, 0]
11, 12	[0, 0, ±0.05]
13, 14	[±0.005, 0, 0]
15, 16	[0, ±0.005, 0]
17, 18	[0, 0, ±0.005]

an action at any of these timesteps. In contrast, a frameskip restricts the agent to select an action at a set interval of frames, with the agent-selected action taken until the next action frame. Implementing a frameskip has been shown to greatly improve the performance of trained agents on the Atari games [41]. The variant of frameskipping used here is that the agent selects an action (moment) to rotate the spacecraft once every 20 timesteps. The agent-selected angular impulse is applied over the next timestep, and free rotation dynamics are propagated for the following 20 timesteps. We found the 20 frameskip value during training to be a reasonable value for the given moment command magnitudes, training time, and control fidelity. The frameskip value (i.e. control frequency) can be decreased during agent implementation, which will generally increase agent control accuracy [13]. The integration fidelity used during the simulation is $\Delta t = 1/240s$, similar to that of popular physics engines such as Bullet [42]. Using a frameskip value of 20 and an integration fidelity of $\Delta t = 1/240$s gives an agent control frequency of $240/21 \approx 11.43$ Hz.

The simulation episode is ended after the agent selects 500 actions (corresponding to real-time of 43.75 seconds) or when the spacecraft angular velocity magnitude exceeds 0.5 rad/s. We implement the neural networks using PyTorch [43]. The hardware used to run train the agents, including Numba acceleration, is left the same as in [13]. This includes all neural network computation done on an NVIDIA RTX 2070 Super GPU.

2.4. Agent Training

For agent training, the policy and value functions were represented as feedforward neural networks with architectures and activation functions given in Appendix A, Table 3. Feedforward neural networks, often called multilayer perceptrons (MLPs), are a paradigm of artificial neural networks in which the data is processed in one direction (forward) without any recurrence. The hyperparameters used in this experiment are also tabulated in Appendix A: Tables 4. The learning rates used and reported are the same for both policy and value networks. Important hyperparameters in this work include the entropy coefficient c and the learning rate. We found that high entropy coefficients (such as 0.001 or 0.01) caused learning instability. We also found that, for the given network architecture, annealing to low learning rates during training was critical for agent success. This is likely due to the numerical fidelity needed in tuning the weights and biases for adequate approximation of the policy function.

The number of layers and neurons of each layer for both policy and value were chosen to be consistent with the architectures used in [13]. The primary agent training differences in this study versus [13] are the use of discrete control instead of continuous control, different training algorithms, and a different reward function. The policy network output layer employs the log softmax function, which converts the logits output by the last hidden layer to the action probabilities respective to the current state. This is much like a typical neural network classification problem, where the policy network "classifies" states to the desired action.

The reward function is likely the most important driver of applied RL performance. The reward is used to formulate the optimization objective for the policy and value functions, thereby directly influencing all aspects of the agent's performance. The reward must numerically correspond to desired agent behavior. Here, the goal for the agent is to slew the spacecraft to the desired orientation and stabilize about that orientation until the end of the episode. This can be formulated as the minimization of the angular error ϕ to zero, as shown in intermediate reward r_a

$$r_a = \begin{cases} \exp\left(\frac{-\phi}{0.14 \cdot 2\pi}\right) & q_{s,t} > q_{s,t-1} \\ \exp\left(\frac{-\phi}{0.14 \cdot 2\pi}\right) - 1 & \text{otherwise} \end{cases} \tag{12}$$

where $q_{s,t}$ 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. The scalar constant of 0.14 in the denominator of the argument is a horizontal scaling factor, chosen so the function returns a magnitude of about 0.001 when $\phi = 180°$. Equation (12) is a crucial extension to

the binary shaped reward in [13]. Our implementation of discrete PPO failed to maximize the reward in [13] successfully. We hypothesize the failure of this agent's maximization of the reward in [13] to be primarily due how exploration is performed in PPO. Exploration in PPO is driven by an entropy term in the loss function, which decreases as the logit distribution narrows. It is likely that the distribution was narrowed when the agent settled in a local extremum, by maximizing the shaped reward before experiencing the terminal reward. This caused the exploration entropy term to decrease, which limited agent exploration near the goal orientation. On-policy algorithms such as PPO discard previous experience, which can cause the agent to "forget" desirable behavior when important experience is discarded. Combining these effects, the agent failed to achieve desirable behavior with simple rewards as in [13]. A continuous reward such as Equation (12) provides much more information to the agent about desirable behavior, allowing on-policy algorithms like PPO to render desirable agent behavior. With an additional bonus if the agent is within the goal criteria ($\phi \leq 0.25°$), the reward given to the agent r_t can be written as

$$r_t = \begin{cases} r_a + 9 & \text{if } \phi \leq 0.25° \\ r_a & \text{otherwise} \end{cases} \tag{13}$$

where r_a is the intermediate reward shown in Equation (12). Simplified, the entire reward formulation can be stated as a negative exponential function, with an added bonus if the agent is within our specified goal criterion of $\phi \leq 0.25°$. This reward function incentivizes the agent to decrease the error angle ϕ as much as possible over consecutive timesteps, thereby agent behavior should approximate time optimality. A further extension of this framework could include an additional penalty for selecting the high-magnitude torques. We find that the negative exponential function gives better agent performance than other functions (such as linear) in reward [17, 44]. We formulate the bonus given in Equation (13) to incentivize the agent to minimize the angular error even when inside the goal angular tolerance. The magnitude of the bonus in Equation (13) was chosen to set the rewards inside the goal about an order of magnitude higher than those in Equation (12). We found this order-of-magnitude difference to be crucial for successful training of the agent.

A terminal reward of +50 is given if the spacecraft terminal state satisifies $\phi \leq 0.25°$, 0 otherwise. A reward of -50 is given if the magnitude of the spacecraft angular velocity exceeds 0.5 rad/s. The agent learns to avoid this behavior due to the large negative reward magnitude.

2.5. Training Results

We trained an agent for 4,340 epochs, ending training when episode reward plateaued near a value of 2500. On the hardware described above, this training run equated to a wall-clock time of about 1 day, 20 hours (with Numba acceleration). Agent statistics are shown in Table 2 for 5,000 simulated episodes, with the agent control frequency set at 40 Hz (frameskip of 5 frames) for statistics calculation. The "closest state" is defined as the state when the error angle ϕ is minimum over the episode. "Terminal state" is defined as the state at the final timestep of the simulated episode.

In the statistics of Table 2, the agent shows excellent success in the simulation environment given the success criteria, as the agent reaches and maintains the desired attitude in all 5,000 evaluation episodes. Comparing the results in Table 2 to those shown in previous work [13], the maximum value of angular error are decreased by two orders of magnitude. These results emphasize that a continuous reward function, such as Equation (13), is crucial in deriving an agent that can achieve pointing accuracies beyond the margin required. The statistics in [13] show agent performance that is barely within the goal angular error tolerance, as the episode return is approximately maximized anywhere inside the given error tolerance. It is desirable to utilize the capabilities of RL to produce an attitude controller that can reach much higher levels of angular accuracy than those simply stated in system requirements, as this agent does. The reward structure in Equations (12) and (13) are shown to be highly effective for this task.

To further compare with work in [13] and demonstrate agent control, we consider three slew maneuvers, tabulated in Appendix B, Table 5. These maneuvers are each a 100° slew about an axis of rotation with equal magnitude components, with the rotation axis being placed in a different octant for each test. The agent's orientation, angular error, and angular velocity histories for the three tests are shown in Appendix B, Figures 5-7. In each test, the agent shows successful performance by constantly decreasing angular error at each timestep, and stabilizing about the goal orientation for the remainder of the episode. The agent pointing accuracies in all agent tests are order of magnitudes better than the set minimum tolerance. The reward formulation in this work allows the derivation of controllers that utilize the strengths and

Table 2. Agent statistics for 5,000 simulated episodes.

	Closest State		Terminal State	
	ϕ (°)	$\|\|\vec{\omega}\|\|$ (rad/s)	ϕ (°)	$\|\|\vec{\omega}\|\|$ (rad/s)
Mean	0.000380	0.001976	0.001954	0.001752
Std. dev.	0.000233	0.001316	0.000705	0.001088
Min	0.000007	0.000066	0.000137	0.000106
Q1	0.000203	0.001387	0.001456	0.001215
Q2	0.000335	0.001868	0.001923	0.001649
Q3	0.000518	0.002371	0.002433	0.002155
Max	0.001980	0.019339	0.004558	0.018723

temporal understanding that a reinforcement learning agent may have over traditional control methods.

3. Conclusions and Future Work

This work presents a reinforcement learning framework for deriving a discrete neural spacecraft attitude controller. Most importantly, we show a reward function and algorithm combination that gives a high degree of pointing accuracy in the developed simulation environment. The reward function used allows the agent to improve pointing accuracy beyond the set minimum accuracy, which could help increase the utility of using reinforcement learning over traditional control methodologies. In the presence of external disturbances or other performance constraints, the ability for the controller to understand long-term process dependencies inherent to reinforcement learning may increase control efficiency and effectiveness. Immediate future work includes using the frameworks presented to implement variations of distributed reinforcement learning. We plan to study how distributed reinforcement learning might be used for both online learning and agents learning from data generated by constellations or swarms of satellites.

References

[1] T. Fong, "Autonomous Systems: NASA Capability Overview." NASA, Report Number ARC-E-DAA-TN62415, October 2018.

[2] C. L. Moore, L. Underwood, F. Figueroa, M. G. Walker, and J. Morris, "NASA Platform for Autonomous Systems (NPAS)." NASA, Report Number HQ-E-DAA-TN70963, 2019.

[3] J. Raiman, S. Zhang, and F. Wolski, "Long-term planning and situational awareness in OpenAI five," *arXiv preprint arXiv:1912.06721*, 2019.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] OpenAI, "OpenAI Five." https://blog.openai.com/openai-five/, 25 June 2018. (accessed 30.09.2020).

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[7] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, and M. Michalski, "Seed rl: Scalable and efficient deep-rl with accelerated central inference," *arXiv preprint arXiv:1910.06591*, 2019.

[8] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *International Conference on Learning Representations*, 2019.

[9] 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.

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

[11] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[12] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, *et al.*, "Deepmind lab," *arXiv preprint arXiv:1612.03801*, 2016.

[13] J. Elkins, R. Sood, and C. Rumpf, "Adaptive continuous control of spacecraft attitude using deep reinforcement learning," in *2020 AAS/AIAA Astrodynamics Specialist Conference*, pp. 20–475, 2020.

[14] D. Izzo, M. Märtens, and B. Pan, "A survey on artificial intelligence trends in spacecraft guidance dynamics and control," *Astrodynamics*, vol. 3, pp. 287–299, 2019.

[15] A. Harris and H. Schaub, "Towards reinforcement learning techniques for spacecraft autonomy," in *42nd Annual AAS Guidance, Navigation and Control Conference*, pp. 18–078, 2018.

[16] A. Harris, T. Teil, and H. Schaub, "Spacecraft decision-making autonomy using deep reinforcement learning," in *29th AAS/AIAA Space Flight Mechanics Meeting, Hawaii*, pp. 1–19, 2019.

[17] B. Gaudet, R. Linares, and R. Furfaro, "Adaptive guidance and integrated navigation with reinforcement meta-learning," *Acta Astronautica*, vol. 169, pp. 180–190, 2020.

[18] B. Gaudet, R. Linares, and R. Furfaro, "Six degree-of-freedom hovering over an asteroid with unknown environmental dynamics via reinforcement learning," in *AIAA Scitech 2020 Forum*, 2020.

[19] J. T. Allison, M. West, A. Ghosh, and F. Vedant, "Reinforcement learning for spacecraft attitude control," in *Proceedings of the International Astronautical Congress*, pp. IAC–19–C1.5.2, International Astronautical Federation, 2019.

[20] Y. Wang, Z. Ma, Y. Yang, Z. Wang, and L. Tang, "A new spacecraft attitude stabilization mechanism using deep reinforcement learning method," in *8th European Conference for Aeronautics and Space Sciences (EUCASS)*, 2019.

[21] 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, pp. 299–309, 2020.

[22] Y. Zhou, E. van Kampen, and Q. P. Chu, "Adaptive spacecraft attitude control with incremental approximate dynamic programming," in *Proceedings of the International Astronautical Congress*, pp. IAC–17–C1.2.5, International Astronautical Federation, 2017.

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

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

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

[26] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[27] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. MIT Press, 1995.

[28] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, pp. 1889–1897, 2015.

[29] *Neural Networks for Machine Learning Lecture 6a: Overview of Mini-Batch Gradient Descent*, 2012.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3803–3810, 2018.

[32] 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,"

in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, IEEE, 2019.

[33] F. L. Markley and J. L. Crassidis, *Fundamentals of Spacecraft Attitude Determination and Control*, vol. 33. Springer, 2014.

[34] A. Bani Younes and D. Mortari, "Derivation of all attitude error governing equations for attitude filtering and control," *Sensors*, vol. 19, no. 21, p. 4682, 2019.

[35] F. L. Markley, "Attitude error representations for kalman filtering," *Journal of Guidance, Control, and Dynamics*, vol. 26, no. 2, pp. 311–317, 2003.

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

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

[38] F. Aghili, "Time-optimal detumbling control of spacecraft," *Journal of Guidance, Control, and Dynamics*, vol. 32, no. 5, pp. 1671–1675, 2009.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[40] Y. Liang, M. C. Machado, E. Talvitie, and M. Bowling, "State of the art control of atari games using shallow reinforcement learning," *arXiv preprint arXiv:1512.01563*, 2015.

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

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

[43] *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019.

[44] B. Gaudet, R. Furfaro, and R. Linares, "A guidance law for terminal phase exo-atmospheric interception against a maneuvering target using angle-only measurements optimized using reinforcement meta-learning," in *AIAA Scitech 2020 Forum*, p. 0609, 2020.

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

Appendix A: Hyperparameters

Table 3. Architectures and activation functions for the policy and value networks used. We used the same as in the original TD3 paper for comparison to previous work [13, 45].

	Policy/Actor Network		Value/Critic Network	
	# of Neurons	Activation Function	# of Neurons	Activation Function
Hidden Layer 1	400	ReLU	400	ReLU
Hidden Layer 2	300	ReLU	300	ReLU
Output Layer	19 (# actions)	LogSoftmax	1	None

Table 4. PPO hyperparameters used for training the agent shown. Arrow indicates linear annealing.

Hyperparameter	Value
Discount (γ)	0.99
Batch size	128
Minibatch size	30
Learning rate	$3 \times 10^{-4} \rightarrow 1 \times 10^{-5}$
Clipping parameter (ϵ)	0.2
Entropy coeff. (c)	0.0001

Appendix B: Agent Control Examples

Table 5. 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]

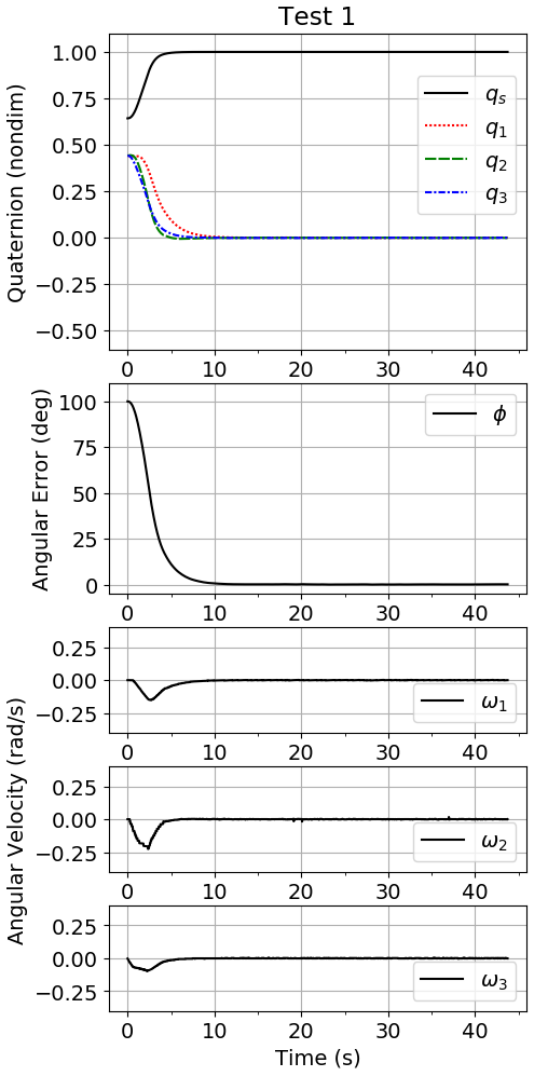

Fig. 5. Agent control history for control example 1, tabulated in Table 5.

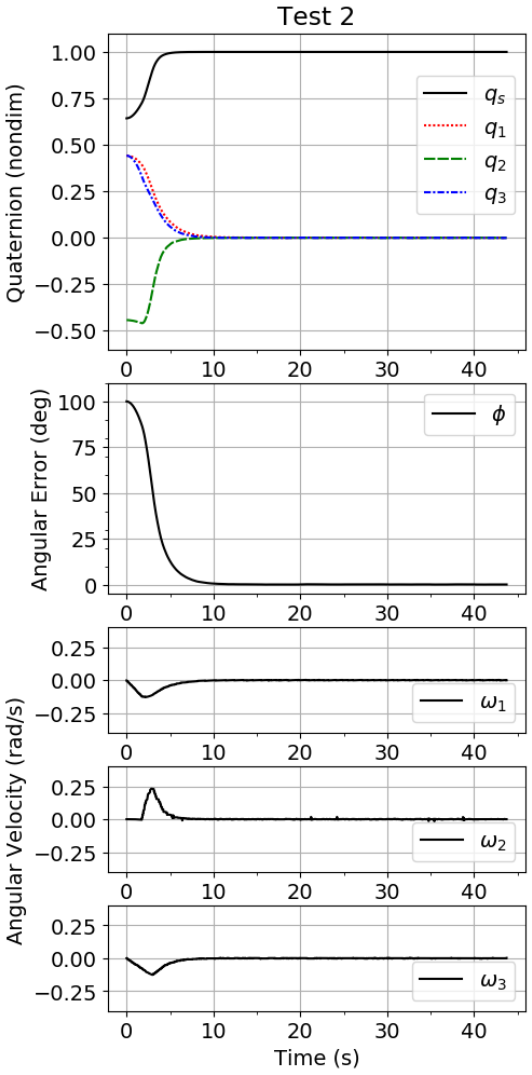

Fig. 6. Agent control history for control example 2, tabulated in Table 5.

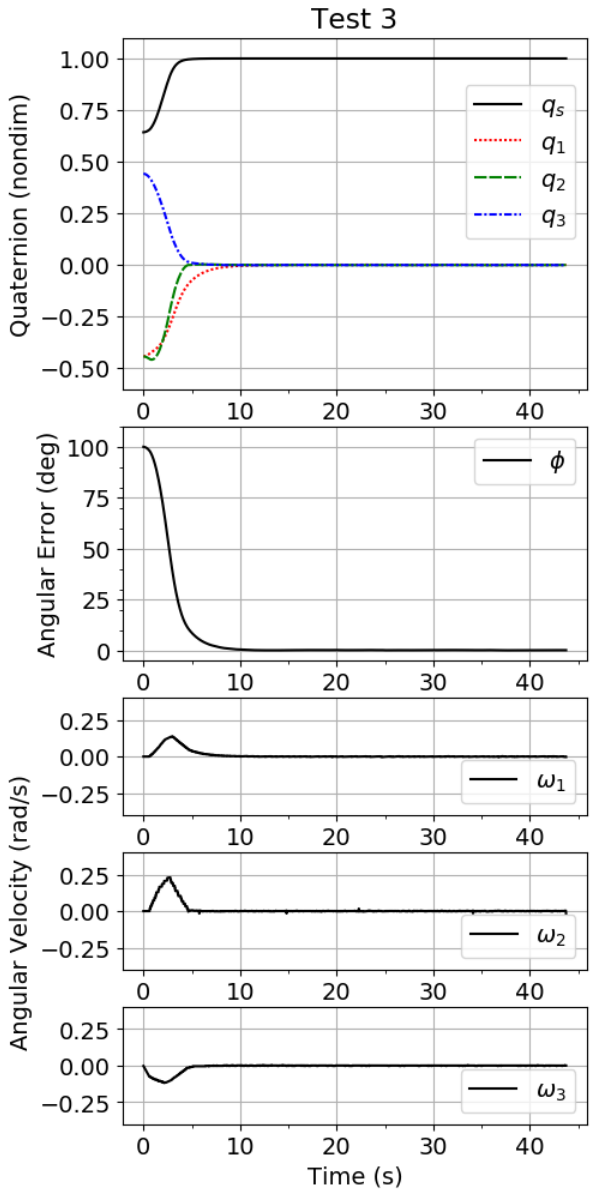

Fig. 7. Agent control history for control example 3, tabulated in Table 5.