Journal of Spacecraft and Rockets
Vol. 58, No. 2, March–April 2021

Deep Reinforcement Learning for Spacecraft Proximity Operations Guidance

Kirk Hovell* and Steve Ulrich†
Carleton University, Ottawa, Ontario K1S 5B6, Canada

https://doi.org/10.2514/1.A34838

This paper introduces a guidance strategy for spacecraft proximity operations, which leverages deep reinforcement learning, a branch of artificial intelligence. This technique enables guidance strategies to be learned rather than designed. The learned guidance strategy feeds velocity commands to a conventional controller to track. Control theory is used alongside deep reinforcement learning to lower the learning burden and facilitate the transfer of the learned behavior from simulation to reality. In this paper, a proof-of-concept spacecraft pose tracking and docking scenario is considered, in simulation and experiment, to test the feasibility of the proposed approach. Results show that such a system can be trained entirely in simulation and transferred to reality with comparable performance.

I. Introduction

AUTONOMOUS spacecraft rendezvous and docking operations have become an active research area in recent decades. Applications, such as on-orbit servicing, assembly, and debris capture [1], require the capability for a chaser spacecraft to autonomously and safely maneuver itself in proximity to a potentially uncooperative target object. A common strategy is pose tracking (i.e., synchronizing the translational and rotational motion of the chaser with respect to the target, such that there is no relative motion between the two objects). Only then does the chaser perform its final approach and capture or dock with the target. Guidance and control algorithms have been developed for this purpose. For example, a guidance and control scheme for capturing a tumbling debris with a robotic manipulator was developed by Aghili [2]. Wilde et al. [3] developed inverse dynamics models to generate guidance paths, and included experimental validation. Ma et al. [4] applied feedforward optimal control for orienting a chaser spacecraft at a constant relative position with respect to a tumbling target. A variety of dual quaternion approaches have been explored [5–7]. Pothen and Ulrich [8,9] used the Udwadia–Kalaba equation to formulate the close-range rendezvous problem, and included experimental validation. Lyapunov vector fields have been used to command docking with an uncooperative target spacecraft by Hough and Ulrich [10].

The currently developed guidance and control techniques presented previously are handcrafted to solve a particular task and require significant engineering effort. As more complex tasks are introduced, the engineering effort needed to handcraft solutions may become infeasible. For example, developing a guidance law for a chaser spacecraft to detumble a piece of spinning space debris, when the two objects are connected via flexible tethers, does not have a clear solution that can be handcrafted. Motivated by more difficult guidance tasks, this paper introduces a new approach that builds upon a branch of artificial intelligence, called deep reinforcement learning, to augment the guidance capabilities of spacecraft for difficult tasks.

Reinforcement learning is based on the idea of an agent trying to choose actions to maximize the rewards it receives over a period of time. The agent uses a policy that, when given an input, returns a suggested action to take. At each time step, a scalar reward, which may be positive or negative, is given to the agent and corresponds to task completion. Through trial and error, the agent attempts to learn a policy that maps inputs to actions, such that the actions taken maximize the rewards received. By selecting an appropriate reward scheme, complex behaviors can be learned by the agent without being explicitly programmed. For a tethered space debris detumbling task, for example, rewards could be given for reducing the angular velocity of the debris, and penalties could be given for fuel usage, forcing the agent to learn a fuel-efficient detumbling policy. The engineering effort is reduced to specifying the reward system rather than the complete logic required to complete the task. This is the main appeal of reinforcement learning. Neural networks have become a popular choice for representing the policy in reinforcement learning, as they are universal function approximators [11]. When neural networks are used within reinforcement learning, the technique is called *deep reinforcement learning*. The core concepts in reinforcement learning have been around for decades, but have only become useful in recent years due to the rapid rise in computing power. Many notable papers have been published recently that use deep reinforcement learning to solve previously unsolvable tasks. In 2015, Mnih et al. [12] applied deep reinforcement learning to play many Atari 2600 [13] games at a superhuman level by training a policy to select button presses (the action) as a function of the screen pixels (the input). Silver et al. [14,15] used deep reinforcement learning to master the game of Go in 2016, a decade earlier than expected.

Training deep reinforcement learning policies on physical robots is time consuming and expensive, and leads to significant wear and tear on the robot because, even with state-of-the-art learning algorithms, the task may have to be repeated hundreds or thousands of times before learning succeeds. Training a policy onboard a spacecraft is not viable due to fuel, time, and computer limitations. An alternative is to train the policy in a simulated environment and transfer the trained policy to a physical robot. If the simulated model is sufficiently accurate and the task is not highly dynamic, policies trained in simulation may be directly transferrable to a real robot [16]. For example, Tai et al. [17] trained a room-navigating robot in simulation and deployed it to reality with success. Although good results were obtained, this technique is unlikely to generalize well to more difficult or dynamic tasks due to the effect known as the simulation-to-reality gap [16,18,19]. This effect states that, because the simulator within which the policy is trained cannot perfectly model the dynamics of the real world, such a policy will fail to perform well in a real-world environment due to overfitting of the simulated dynamics. Efforts to get around this problem often take advantage of domain randomization [20] (i.e., randomizing the environmental parameters for each simulation to force the policy to become robust to environmental changes). Domain randomization has been used successfully in a drone racing task [21] and in the manipulation of objects with a robotic hand [20]. However, domain randomization significantly increases the training time because the policy must learn to become

Presented as Paper 2020-1600 at the AIAA Scitech 2020 Forum, Orlando, FL, January 6–10, 2020; received 21 April 2020; accepted for publication 22 October 2020; published online 14 January 2021. Copyright © 2020 by The Authors. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. All requests for copying and permission to reprint should be submitted to CCC at www.copyright.com; employ the eISSN 1533-6794 to initiate your request. See also AIAA Rights and Permissions www.aiaa.org/randp.

*Ph.D. Candidate, Department of Mechanical and Aerospace Engineering, 1125 Colonel By Drive. Student Member AIAA.

†Associate Professor, Department of Mechanical and Aerospace Engineering, 1125 Colonel By Drive. Senior Member AIAA.

adept at all dynamic variations of the task. Other efforts to solve the simulation-to-reality problem involve continuing to train the policy once deployed to experiment. A drifting car [22] policy was partially trained in simulation, and then fine-tuning training was performed once experimental data were collected. A quadrotor stabilization task [23] summed the policy output with a conventional proportional derivative controller to guide the learning process and help with the simulation-to-reality transfer.

Deep reinforcement learning has also been applied to aerospace applications, although mostly in simulation. A simulated fleet of wildfire surveillance aircraft used deep reinforcement learning to command the flight path of the aircraft [24]. Deep reinforcement learning has also been applied to spacecraft map generation while orbiting small bodies [25], spacecraft orbit control in unknown gravitational fields [26], and spacecraft orbital transfers [27]. Others have used reinforcement learning to train a policy that performs guidance and control for pinpoint planetary landing [28,29]. Neural networks have been trained to approximate off-line-generated optimal guidance paths for pinpoint planetary landing, such that the neural network approximates an optimal guidance algorithm that can be executed in real time [30,31].

Inspired by the ability of deep reinforcement learning to have behavior be learned rather than handcrafted, and motivated by the need for new simulation-to-reality transfer techniques, this paper introduces a novel technique that allows for the use of reinforcement learning on a real spacecraft platform. The proposed technique builds off of the planetary landing work [28,29], where the neural networks were trained to approximate handcrafted optimal guidance trajectories and a conventional controller was used to track the approximated trajectory. Here, deep reinforcement learning is used to train a guidance policy whose trajectories are fed to a conventional controller to track. Using reinforcement learning allows an unbiased guidance policy to be discovered by the agent instead of being shown many handcrafted guidance trajectories to mimic. The proposed technique is in contrast to typical reinforcement learning research, where the policy is responsible for learning both the guidance and control logic. By restricting the policy to learn only the guidance portion, we harness the high-level, unbiased, task-solving abilities of reinforcement learning while deferring the control aspect to the well-established control theory community. Control theories have been developed that are able to perform trajectory tracking well under dynamic uncertainty [32–34], and can therefore handle model discrepancies between simulation and reality. Harris et al. [35] recently presented a strategy that uses reinforcement learning to switch between a set of available controllers depending on the system state. The authors [35] suggest that reinforcement learning should not be tasked with learning guidance and control because control theory already has great success. By combining deep reinforcement learning for guidance with conventional control theory, the policy is prevented from learning a controller that overfits the error-prone simulated dynamics. We call our deep reinforcement learning guidance strategy *deep guidance*. The novel contributions of this work are 1) the deep guidance technique, which combines deep reinforcement learning as guidance with a conventional controller; 2) experimental demonstrations showing that this deep guidance strategy can be trained in simulation and deployed to reality without any fine-tuning; and 3) the first, to the best of the authors' knowledge, experimental demonstration of artificial intelligence commanding the motion of a spacecraft platform.

It should be noted that, although the authors were motivated by difficult guidance tasks, such as detumbling tethered space debris, in this paper, a proof-of-concept task—a simple spacecraft pose tracking and docking scenario—is considered. Demonstrating the deep guidance technique on a simple task will highlight its potential for use on more difficult tasks.

This paper is organized as follows: Sec. II presents background on deep reinforcement learning and the specific learning algorithm used in this paper, Sec. III presents the novel guidance concept developed by the authors, Sec. IV describes the pose tracking scenario considered, Sec. V presents numerical simulations demonstrating the effectiveness of the technique, Sec. VI presents the experimental results, and Sec. VII concludes this paper.

II. Deep Reinforcement Learning

The goal of deep reinforcement learning is to discover a policy, π_θ, represented by a feedforward neural network and whose subscript denotes it has trainable weights θ that maps states, $\boldsymbol{x} \in \mathcal{X}$, to actions, $\boldsymbol{a} \in \mathcal{A}$ that, when executed, maximize the expected rewards received over one episode. (In reinforcement learning, one simulation is called one episode.) The action is obtained by feeding the state to the policy, as follows:

$$\boldsymbol{a} = \pi_\theta(\boldsymbol{x}) \tag{1}$$

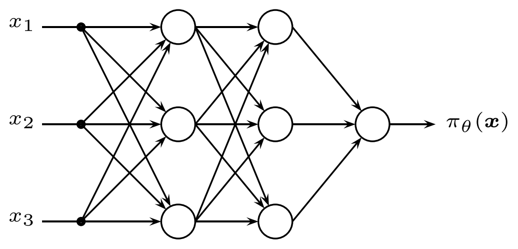
Although many deep reinforcement learning algorithms are currently available, this work uses the distributed distributional deep deterministic policy gradient (D4PG) algorithm [36]. The D4PG algorithm, released in early 2018, was selected because it operates in continuous state and action spaces, it has a deterministic output, it can be trained in a distributed manner to take advantage of multi-CPU machines, and it achieves state-of-the-art performance.

The D4PG [36] algorithm is an actor–critic algorithm, implying there is a policy neural network, $\pi_\theta(\boldsymbol{x})$, that maps states to actions and a value neural network, $Z_\phi(\boldsymbol{x}, \boldsymbol{a})$, with trainable weights ϕ that maps a state–action pair to a probability distribution of the predicted discounted rewards for the remainder of the episode. The total discounted rewards expected to be received from a given state, $\boldsymbol{x}$, and taking action $\boldsymbol{a}$ from the policy, π_θ, is given by

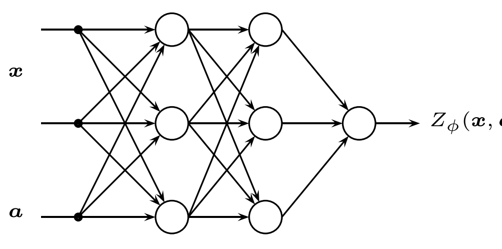
$$J(\theta) = \mathbb{E}\{Z_\phi(\boldsymbol{x}, \pi_\theta(\boldsymbol{x}))\} \tag{2}$$

where $J(\theta)$ is the expected reward from the given state as a function of the policy weights θ, and $\mathbb{E}$ denotes the expectation. The objective of reinforcement learning is then to find a policy π_θ that maximizes $J(\theta)$ through systematically adjusting θ.

To maximize Eq. (2), we must first establish the policy and value neural networks, shown in Fig. 1. The policy network, shown in Fig. 1a, accepts the system state $\boldsymbol{x}$ as the input, shown with three elements x_1, x_2, and x_3 in the figure, and outputs a commanded action, $\boldsymbol{a} = \pi_\theta(\boldsymbol{x})$, which may be multidimensional. Each arrow in the network corresponds to a trainable weight that is parameterized by θ. The value network, shown in Fig. 1b, accepts the state and



a) Policy neural network



b) Value neural network

Fig. 1 Policy and value neural networks used in the D4PG algorithm.

action vectors as inputs, and outputs a value distribution, $Z_\phi(\boldsymbol{x}, \boldsymbol{a})$. In other words, it returns a probability distribution of how many discounted rewards are predicted to be received from taking action $\boldsymbol{a}$ in state $\boldsymbol{x}$ and continuing to follow the policy until the end of the episode.

Simulated state, action, next state, and reward data are obtained from running episodes and placing the data in a large replay buffer that stores the most recent R data. Batches of size M of simulated data are randomly sampled from the replay buffer and used to train the policy and value neural networks. From a given batch of state, action, next state, and reward data, the value network can be trained using supervised learning. Gradient descent is used to minimize the cross-entropy loss function given by

$$L(\phi) = \mathbb{E}[-Y \log(Z_\phi(\boldsymbol{x}, \boldsymbol{a}))] \tag{3}$$

where Y is the target value distribution (i.e., the new best prediction of the true value distribution based on the sampled data), as first introduced by Bellmare et al. [37]. It is calculated using

$$Y = \sum_{n=0}^{N-1} \gamma^n r_n + \gamma^N Z_{\phi'}(\boldsymbol{x}_N, \pi_{\theta'}(\boldsymbol{x}_N)) \tag{4}$$

where r_n is the reward received at time step n, γ is the discount factor (to weigh current rewards higher than future rewards), and $Z_{\phi'}(\boldsymbol{x}_N, \pi_{\theta'}(\boldsymbol{x}_N))$ is the value distribution evaluated N time steps into the future. The N-step return [38] is used, where N data points into the future are included for a more accurate prediction of Y. It should also be noted that the symbols θ' and ϕ' indicate that these are not the true weights of the policy and value networks, but rather an exponential moving average of them, calculated by

$$\theta' = (1 - \epsilon)\theta' + \epsilon\theta \tag{5}$$

$$\phi' = (1 - \epsilon)\phi' + \epsilon\phi \tag{6}$$

with $\epsilon \ll 1$. Having a copy of the policy and value networks with smoothed weights has been empirically shown to have a stabilizing effect on the learning process [12].

Equation (4) recursively calculates an updated prediction for the value distribution for a given state–action pair according to the reward data received. Then, by minimizing the loss function in Eq. (3) through adjusting ϕ, using learning rate β, the value network slowly approaches these updated predictions, which are then smoothed and used in Eq. (4). This recursive process led Sutton and Barto [39] to write, "we learn a guess from a guess."

With the training procedure for the value network outlined, the policy network must now be trained. The goal is to adjust the policy parameters θ in the direction of increasing the expected discounted rewards for a given state, $J(\theta)$. Because neural networks are differentiable, the chain rule can be used to compute:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial J(\theta)}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \theta} \tag{7}$$

where $\partial J(\theta)/\partial \boldsymbol{a}$ is computed from the value network, and $\partial \boldsymbol{a}/\partial \theta$ is computed from the policy network. In a sense, we differentiate through the value network into the policy network. More formally

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \pi_\theta(\boldsymbol{x}) \mathbb{E}[\nabla_{\boldsymbol{a}} Z_\phi(\boldsymbol{x}, \boldsymbol{a})]|_{\boldsymbol{a}=\pi_\theta(\boldsymbol{x})}] \tag{8}$$

describes how the policy parameters θ should be updated to increase the expected rewards when the policy π_θ is used. Finally, the parameters θ are updated via

$$\theta = \theta + \nabla_\theta J(\theta)\alpha \tag{9}$$

for a learning rate α.

To implement the algorithm, K agents run independent episodes using the most up-to-date version of the policy. Gaussian exploration noise is applied to the chosen action to force exploration and is the basis for discovering new strategies. The action is obtained through

$$\boldsymbol{a}_t = \pi_\theta(\boldsymbol{x}_t) + \mathcal{N}(0, \sigma^2) \tag{10}$$

where $\mathcal{N}(0, \sigma^2)$ is the normal distribution with a mean of 0 and an exploration noise standard deviation of σ. The collected data are the state $\boldsymbol{x}_t$, action $\boldsymbol{a}_t$, reward r_t, and next state $\boldsymbol{x}_{t+N}$, and are placed into a replay buffer that stores the most recent R data points. Asynchronously, a learner randomly samples batches of data from the replay buffer and uses them to train the value network one step using Eq. (3), and then trains the policy network one step using Eq. (9). Over time, the accuracy of the value network and the average performance of each agent are expected to increase.

III. Deep Guidance

Using deep reinforcement learning for spacecraft guidance may allow for difficult tasks to be accomplished through learning an appropriate behavior rather than handcrafting such a behavior. In order for the reinforcement learning algorithm presented in Sec. II to be trained in simulation and deployed to reality without any fine-tuning on the spacecraft, it is proposed that the learning algorithm cannot be responsible for the entire guidance, navigation, and control (GNC) stack. This is to prevent the policy from overfitting the simulated dynamics and being unable to handle the transition to a dynamically uncertain real world (i.e., the simulation-to-reality problem) [16]. For this reason, the authors present a system, called deep guidance, which uses deep reinforcement learning as a guidance system along with a conventional controller. Conventional controllers are able to handle dynamic uncertainties and modeling errors that typically plague reinforcement learning policies that attempt to learn the entire GNC routine. It is assumed that perfect navigation is available. A block-scheme diagram of the proposed system is shown in Fig. 2. The learned deep guidance block has the current state $\boldsymbol{x}_t$ as its input and the desired velocity $\boldsymbol{v}_t$ as its output. The desired velocity is fed to a conventional controller, which also receives the current state $\boldsymbol{x}_t$ and calculates a control effort $\boldsymbol{u}_t$. The control effort is executed on the dynamics that generate a scalar reward r_t and the next state $\boldsymbol{x}_{t+1}$.

During training of the deep guidance model, an ideal controller is assumed, as shown in Fig. 3. This ensures that the guidance model does not overfit to any specific controller, thereby making this approach controller-independent. Because the ideal controller perfectly commands the dynamics to move at the desired velocity $\boldsymbol{v}_t$, the ideal controller and the dynamics model may be combined into a single kinematics model.

Once trained, any controller may be used alongside the deep guidance system for use on a real robot. Because the deep guidance system only experiences an ideal controller during training, it is possible that the guidance model will not experience any nonideal states that may be encountered when using a real controller, which may harm performance. For this reason, Gaussian noise may be applied to the output of the kinematics to force the system into undesirable states during training that may be encountered by a nonideal controller.

The proposed system is trained and tested on a spacecraft pose tracking and docking task detailed in the following section.

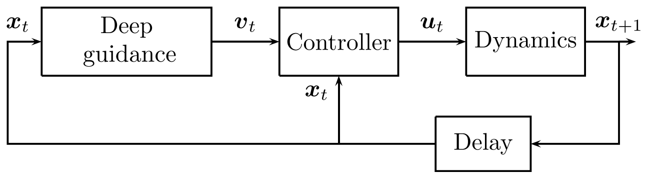


Fig. 2 Proposed deep guidance strategy.

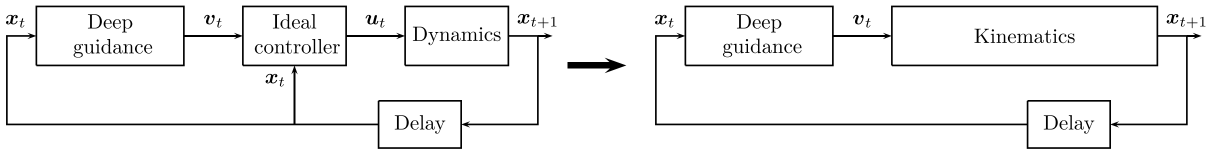


Fig. 3 Proposed deep guidance with an ideal controller for training purposes in simulation.

IV. Problem Statement

This section presents the simulated spacecraft pose tracking and docking environment the deep guidance system will be trained within. Although the deep guidance technique may allow for more complex behaviors to be learned, a simple task is considered here as a proof of concept. Similarly to other spacecraft researchers with experimental validation [3,40–42], a double-integrator planar dynamics model is used such that the available experimental facility can validate the simulated results. Including a full orbital model was deemed to be outside the scope of this paper, because the goal of this work was to demonstrate the simulation-to-reality ability of the proposed deep guidance approach. Furthermore, a double-integrator model is representative of proximity operations in orbit over small distances and timescales [43].

A chaser spacecraft exists in a planar laboratory environment, and it is tasked with approaching and docking with a target spacecraft, as shown in Fig. 4. The chaser and target spacecraft start at rest. The chaser spacecraft is given some time to maneuver to the hold point in front of the target. Then, the chaser spacecraft is tasked with approaching the target such that the two may dock.

A. Kinematics and Dynamics Models

During training, a kinematics model is used, which approximates a dynamics model and an ideal controller, as shown in Fig. 3. The deep guidance policy accepts the chaser state error $\boldsymbol{e}_t$ and outputs the commanded action, which in this case is the velocity $\boldsymbol{v}_t$:

$$\boldsymbol{x}_t = [x \quad y \quad \psi]^T \tag{11}$$

$$\boldsymbol{e}_t = \boldsymbol{x}_d - \boldsymbol{x}_t \tag{12}$$

$$\boldsymbol{v}_t = \pi_\theta(\boldsymbol{e}_t) \tag{13}$$

where $\boldsymbol{x}_d$ is the desired state, and $\boldsymbol{x}_t$ is the state of the system, where x and y represent the X and Y locations of the chaser, respectively, and ψ represents the orientation of the chaser. The velocity $\boldsymbol{v}_t$ is numerically integrated using the SciPy [44] Adams/backward differentiation formula methods in Python to obtain $\boldsymbol{x}_{t+1}$.

To evaluate the learning performance, the trained policy is periodically evaluated on an environment with full dynamics and a controller, as shown in Fig. 2. In other words, it is "deployed" to another simulation for evaluation in much the same way that it will be deployed to an experiment in Sec. VI. The deep guidance policy outputs the desired velocity as in Eq. (13) that is fed to a simple proportional velocity controller of the form:

$$\boldsymbol{u}_t = \boldsymbol{K}_p(\boldsymbol{v}_t - \dot{\boldsymbol{x}}_t) \tag{14}$$

where $\boldsymbol{K}_p = \mathrm{diag}\{2, 2, 0.1\}$, and $\boldsymbol{u}_t$ is the control effort. The $\boldsymbol{K}_p$ values were chosen by trial and error until satisfactory performance was achieved.

A double-integrator dynamics model is used to simulate the motion of the chaser:

$$\ddot{x} = \frac{F_x}{m} \tag{15}$$

$$\ddot{y} = \frac{F_y}{m} \tag{16}$$

$$\ddot{\psi} = \frac{\tau}{I} \tag{17}$$

where F_x and F_y are the forces applied in the X and Y directions, respectively; τ is the torque applied about the Z axis; m is the chaser spacecraft mass; I is its moment of inertia; $\ddot{x}$ is the acceleration in X; $\ddot{y}$ is its acceleration in Y; and $\ddot{\psi}$ is the angular acceleration about Z. The accelerations are integrated twice to obtain the position and orientation at the following time step.

The following subsection discusses the reward function used to incentivize the desired behavior.

B. Reward Function

To calculate the reward given to the agent at each time step, a reward field $\boldsymbol{f}$ is generated according to

$$\boldsymbol{f}(\boldsymbol{x}_t) = -|\boldsymbol{e}_t| \tag{18}$$

The reward field is zero at the desired state and becomes negative linearly as the chaser moves away from the desired state.

The reward given to the agent, $r_t(\boldsymbol{x}_t, \boldsymbol{a}_t)$, depends on the action taken in a given state. Therefore, the difference in the reward field between the current and previous time steps is used to calculate the reward given to the agent:

$$r_t = \|\boldsymbol{K}(\boldsymbol{f}(\boldsymbol{x}_t) - \boldsymbol{f}(\boldsymbol{x}_{t-1}))\| \tag{19}$$

The states are weighted with $\boldsymbol{K} = \mathrm{diag}\{0.5, 0.5, 0.1\}$ to ensure the rotational component does not dominate the reward field. By rewarding the change in reward field, a positive reward will be given to the agent if it chooses an action that moves the chaser closer to the desired state and a negative reward otherwise. Two additional penalties are included to encourage the desired behavior: a velocity-error penalty and a collision penalty. To avoid chaser oscillations, velocity errors are penalized near the desired state. To avoid collisions, the reward is reduced by $r_{\mathrm{collide}} = 15$ when the chaser collides with the target:

$$r_t = \begin{cases} \|\boldsymbol{K}(\boldsymbol{f}(\boldsymbol{x}_t) - \boldsymbol{f}(\boldsymbol{x}_{t-1}))\| - c_1 \dfrac{\|\boldsymbol{v}_t - \boldsymbol{v}_{\mathrm{ref}}\|}{\|\boldsymbol{e}_t\| + \eta} - r_{\mathrm{collide}} & \text{for } \|\boldsymbol{d}_t\| \le 0.3 \\ \|\boldsymbol{K}(\boldsymbol{f}(\boldsymbol{x}_t) - \boldsymbol{f}(\boldsymbol{x}_{t-1}))\| - c_1 \dfrac{\|\boldsymbol{v}_t - \boldsymbol{v}_{\mathrm{ref}}\|}{\|\boldsymbol{e}_t\| + \eta} & \text{otherwise} \end{cases} \tag{20}$$

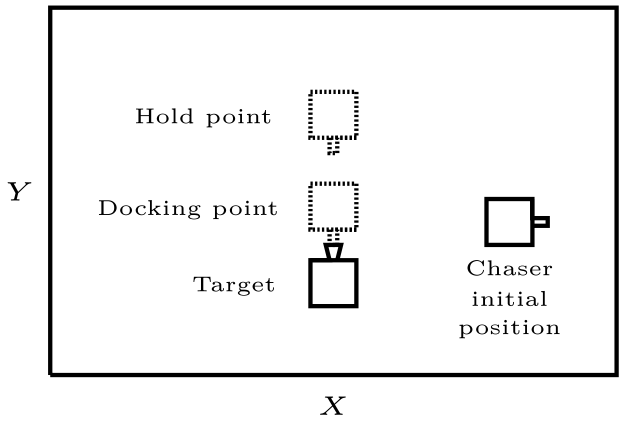
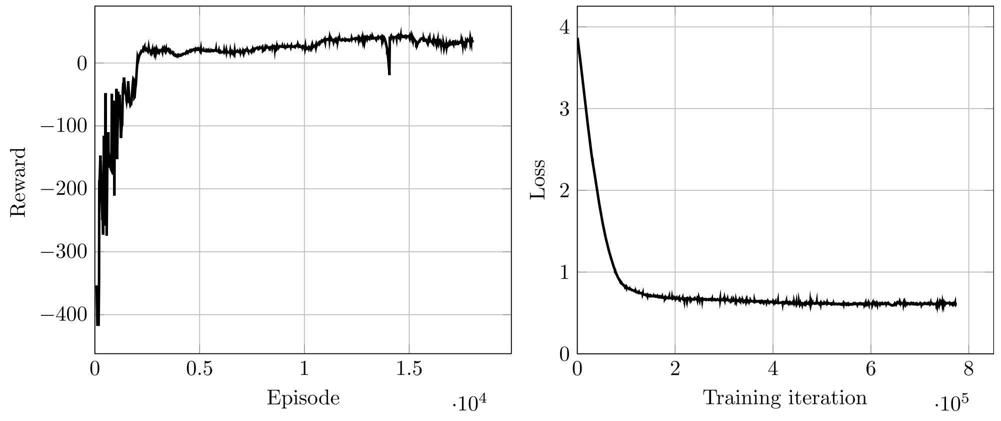


Fig. 4 Spacecraft pose tracking and docking task.



a) Learning curve b) Loss

Fig. 5 Training performance for chaser pose tracking and docking for a stationary target with constant initial conditions.

Here, $\boldsymbol{v}_t$ is the chaser velocity, $\boldsymbol{v}_{\mathrm{ref}}$ is the velocity of the hold/docking point, $\eta = 0.01$ is a small constant, $c_1 = 0.5$ is used to weigh the velocity penalty such that it does not dominate the reward function, and $\boldsymbol{d}_t$ is the distance between the chaser and the target.

The learning algorithm details are presented in the following subsection.

C. Learning Algorithm Details

The policy and the value neural networks have 400 neurons in their first hidden layer and 300 neurons in their second layer; Fig. 1 is not to scale. In the value network, the action is fed directly into the second layer of the network, as this was empirically shown to be beneficial [36,45]. The value network therefore has 138,951 trainable parameters ϕ and the policy network has 123,603 trainable parameters θ. Each neuron in both hidden layers uses a rectified linear unit as its nonlinear activation function, shown as follows:

$$g(y) = \begin{cases} 0 & \text{for } y < 0 \\ y & \text{for } y \geq 0 \end{cases} \tag{21}$$

In the output layer of the policy network, a $g(y) = \tanh(y)$ nonlinear activation function is used to force the commanded velocity to be bounded. The output layer of the value neural network uses the softmax function, shown as follows, to force the output value distribution to indeed be a valid probability distribution:

$$g(y_i) = \frac{e^{y_i}}{\sum_{k=1}^{K} e^{y_k}} \quad \forall\ i = 1, \ldots, B \tag{22}$$

for each element y_i and for B bins in the value distribution. Drawing from the original value distribution paper [37], $B = 51$ bins are used in this work. The value distribution bins are evenly spaced on the empirically determined interval $[-1000, 100]$, as this is the range of accumulated rewards encountered during this pose tracking and docking task.

The policy and value networks are trained using the Adam stochastic optimization routine [46] with a learning rate of $\alpha = \beta = 0.0001$. The replay buffer R can contain 10^6 transition data points, and a mini batch size $M = 256$ is used. The smoothed network parameters are updated on each training iteration with $\epsilon = 0.001$. The noise standard deviation applied to the action to force exploration during training is $\sigma = (1/3)[\max(\boldsymbol{a}) - \min(\boldsymbol{a})](0.9999)^E$, where E is the episode number; having a standard deviation of one-third the action range empirically leads to good exploration of the action space. The noise standard deviation is decayed exponentially as more episodes are performed to narrow the action search area, at a rate that halves roughly every 7000 episodes. Ten actors are used, $K = 10$, such that simulated data using the most up-to-date version of the policy are being collected by 10 actors simultaneously. A discount factor of $\gamma = 0.99$ was used along with an N-step return length of $N = 1$. The TensorFlow[‡] machine learning framework was used to generate and train the neural networks.

Every five training episodes, the current policy is deployed and run in a full dynamics environment with the proportional velocity controller in Eq. (14) to evaluate its performance, as shown in Fig. 2. During deployment, $\sigma = 0$ in Eq. (10) such that no exploration noise is applied to the deep guidance velocity commands.

V. Simulation Results

To test the deep guidance approach, three variations of the spacecraft pose tracking and docking task are studied. The first uses a stationary target, the second uses a rotating target, and the third uses a rotating target with a stationary obstacle that must be avoided. The first scenario is run both with constant initial conditions and with randomized initial conditions, whereas the second and third scenarios use randomized initial conditions. The 30 cm cube spacecraft platform has a uniform simulated mass of 10 kg.

A. Docking with a Stationary Target

The chaser and target nominal initial conditions are [3 m, 1 m, 0 rad] and [1.85 m, 0.6 m, $(\pi/2)$ rad] for the chaser and target, respectively. The hold point is 1.0 m offset from the front face of the target, and the docking point is 0.5 m offset. Each episode is run for 90 s with a 0.2 s time step. For the first 45 s, the desired state is the hold point, and afterward, it is the docking point. The commanded velocity bounds are ± 0.05 m/s and $\pm(\pi/18)$ rad/s.

Results of the first spacecraft pose tracking and docking task are shown in Fig. 5, and are the results of 47 h of training on an Intel® i7-8700 K CPU. The learning curve, shown in Fig. 5a, plots the total rewards received on each episode, which increase, as expected, during training. The deep guidance strategy successfully learned the desired behavior after roughly 11,000 episodes. The loss function, calculated using Eq. (3) and shown in Fig. 5b, decreases as anticipated, indicating that the value-network output distribution is approaching the target values calculated using Eq. (4), on average. Sample trajectories during training are shown in Fig. 6. The gray object represents the initial pose of the chaser, the dashed line represents its trajectory, and the solid black object represents its final pose. The pose tracking and docking task was successfully learned.

Next, the chaser and target initial conditions were randomized at the beginning of each episode to force the deep guidance system to generalize across a range of initial states and not simply master a single trajectory. The initial states were randomized around the nominal ones according to a normal distribution with a standard deviation of 0.3 m for position and $(\pi/2)$ rad for attitude. Figure 7 shows the learning curve and associated loss function during training. The learning curve, shown in Fig. 7a, shows that the agent success-

[‡]Software available from https://www.tensorflow.org/.

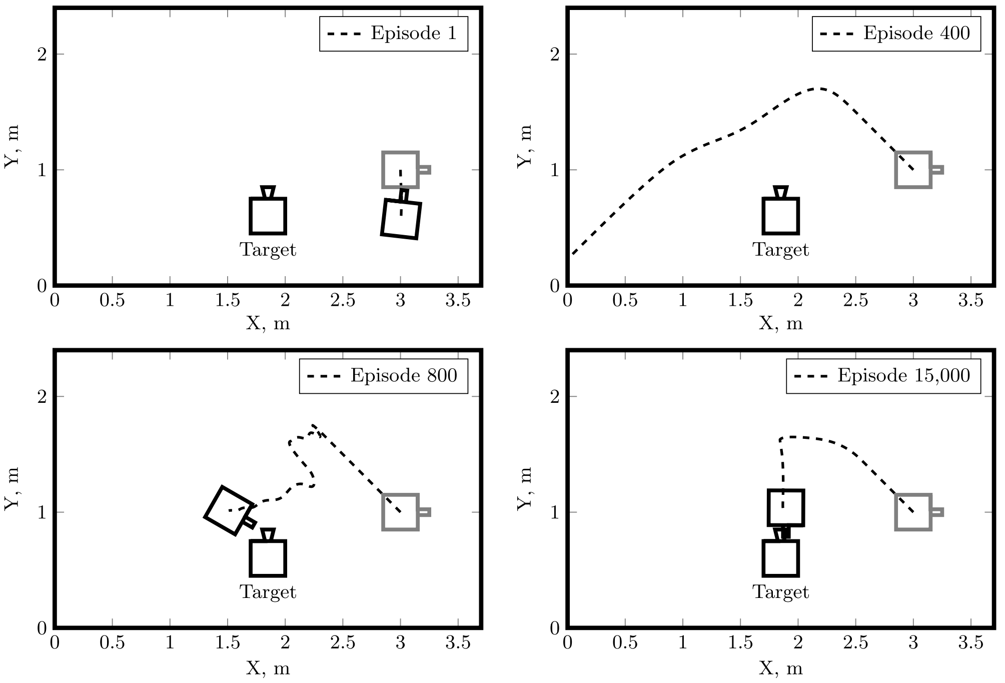


Fig. 6 Visualization of chaser trajectories at various episodes during the training process.

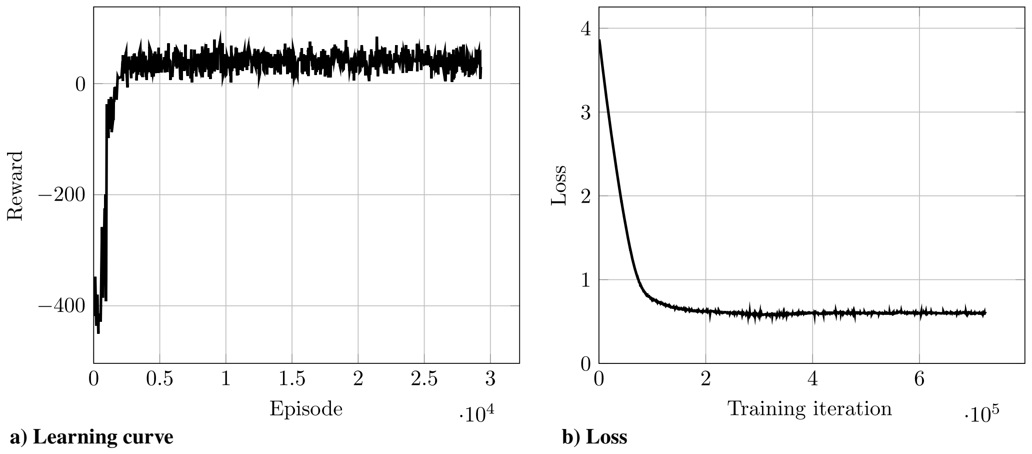


Fig. 7 Training performance for chaser pose tracking and docking for a stationary target with randomized initial conditions.

fully learned a more general guidance strategy in the presence of randomized initial conditions. The learning curve appears noisier than the learning curve shown in Fig. 5a because each episode may have slightly more or less rewards available depending on the initial conditions. Sample trajectories once training was complete are shown in Fig. 8.

B. Docking with a Spinning Target

This subsection presents the second scenario that the deep guidance system was trained on, that is, a spacecraft pose tracking and docking task in the presence of a spinning target. All learning parameters are identical to those presented in Sec. IV.C, demonstrating the generality of the proposed deep guidance approach. The target spacecraft is given a constant counterclockwise angular velocity of $\omega = (\pi/45)$ rad/s. The velocity bounds on the chaser are increased to ± 0.1 m/s. Each episode is run for 180 s with a 0.2 s time step. For the first 90 s, the chaser is incentivized to track the moving hold position, and afterward, it is rewarded for tracking the moving docking point. The initial conditions have a mean of [3 m, 1 m, 0 rad] and [1.85 m, 1.2 m, 0 rad] for the chaser and target, respectively, and a standard deviation of 0.3 m for position and $(\pi/2)$ rad for attitude. Because the target is rotating, the hold and docking points are inertially moving with time. The learning curve in Fig. 9a shows that a deep guidance policy was successfully learned. Sample trajectories, shown in Fig. 10, show example motion of the chaser. The target performs two complete rotations during the episode so that its initial and final orientations are as shown.

C. Docking While Avoiding an Obstacle

This section presents a numerical simulation that is identical to Sec. V.B except for the addition of a stationary obstacle that must be avoided. The input to the policy is modified to include the distance from the chaser to the obstacle, as follows:

$$\boldsymbol{o}_t = [\boldsymbol{e}_t^T \quad \boldsymbol{d}_t^T]^T \tag{23}$$

where $\boldsymbol{d}_t$ are the X and Y distances from the chaser to the obstacle, respectively. For this scenario, the deep guidance velocity is calculated through

$$\boldsymbol{v}_t = \pi_\theta(\boldsymbol{o}_t) \tag{24}$$

The r_{collide} penalty is also applied when the chaser collides with the target. Collision occurs when the center of mass of the chaser is within 0.3 m of the target. The position of the target is [1.2, 1.2] m.

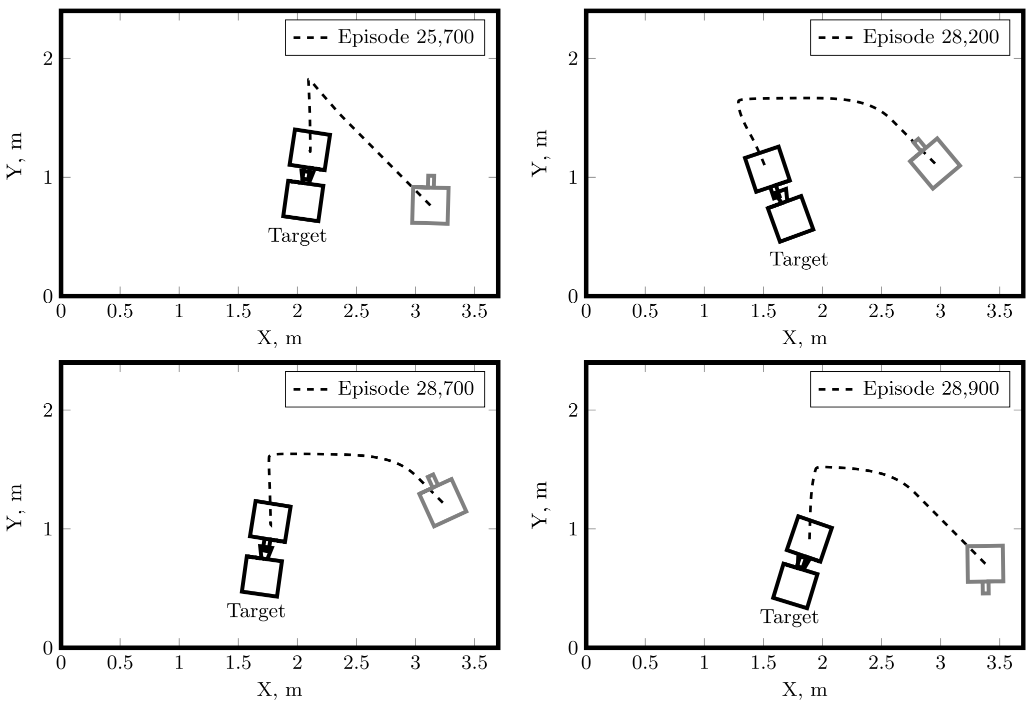


Fig. 8 Examples of learned chaser trajectories with randomized initial conditions.

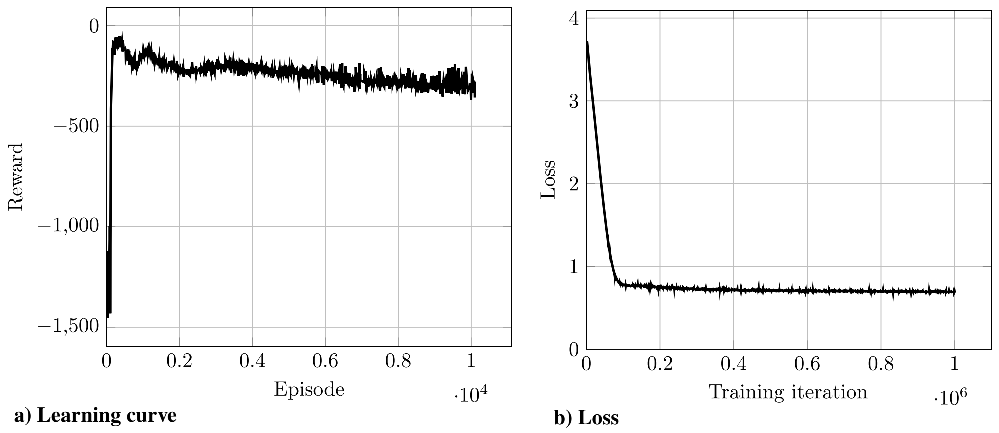


Fig. 9 Training performance for chaser pose tracking and docking with a rotating target.

The learning curve in Fig. 11a shows that a deep guidance policy was successfully learned. Sample trajectories, shown in Fig. 12, show the motion of the chaser. A policy that causes the chaser to track the target while avoiding collision with the obstacle was successfully learned.

The deep guidance system presented in this paper was successfully trained on simulated spacecraft pose tracking and docking tasks. It allows the designer to easily specify a reward function to convey the desired behavior to the agent instead of handcrafting a guidance trajectory. Although the guidance trajectories learned in this paper by the deep guidance system would be trivial to handcraft, the purpose of this paper is to introduce the deep guidance technique. It is expected that the deep guidance technique will unlock the ability for more difficult learned guidance strategies in future work. All codes used are open source and can be accessed at http://www.github.com/Kirkados/JSR2020_D4PG.

The trained guidance policies are tested in experiment in the following section.

VI. Experimental Validation

To validate the numerical simulations and to test if the proposed deep guidance strategy can overcome the simulation-to-reality gap, experiments are performed in a laboratory environment at Carleton University. A planar gravity-offset testbed is used, where two spacecraft platforms are positioned on a flat granite surface. Air bearings are used to provide a near-friction-free planar environment. The experimental facility is discussed, followed by the experimental setup and results.

A. Experiment Facility

Experiments were conducted at the Spacecraft Robotics and Control Laboratory of Carleton University, using the Spacecraft Proximity Operations Testbed (SPOT). Specifically, SPOT consists of two air-bearing spacecraft platforms operating in close proximity on a 2.4×3.5 m granite surface. The use of air bearings on the platforms reduces the friction to a negligible level. Because of surface slope angles of 0.0026 and 0.0031 deg along both directions, residual gravitational accelerations of 0.439 and 0.525 $\mathrm{mm/s^2}$ perturb the dynamics of the floating platforms along the X and Y directions, respectively. Both platforms have dimensions of $0.3 \times 0.3 \times 0.3$ m, and are actuated by expelling compressed air at 550 kPa (80 psi) through eight miniature air nozzles distributed around each platform, thereby providing full planar control authority. Each thruster generates approximately 0.25 N of thrust and is controlled at a frequency of

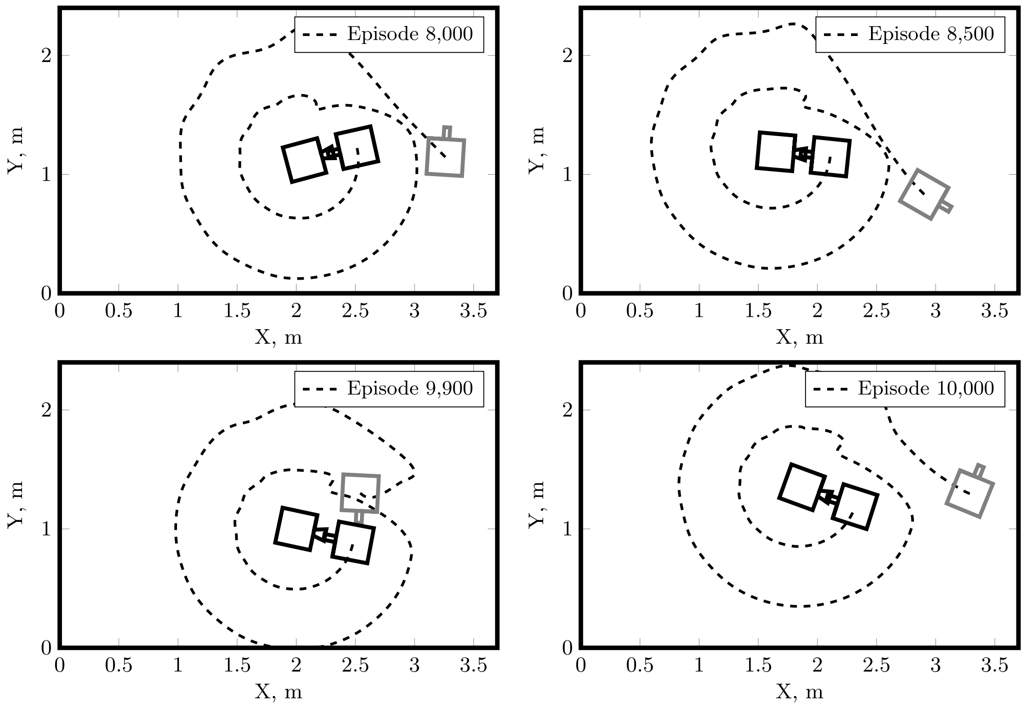
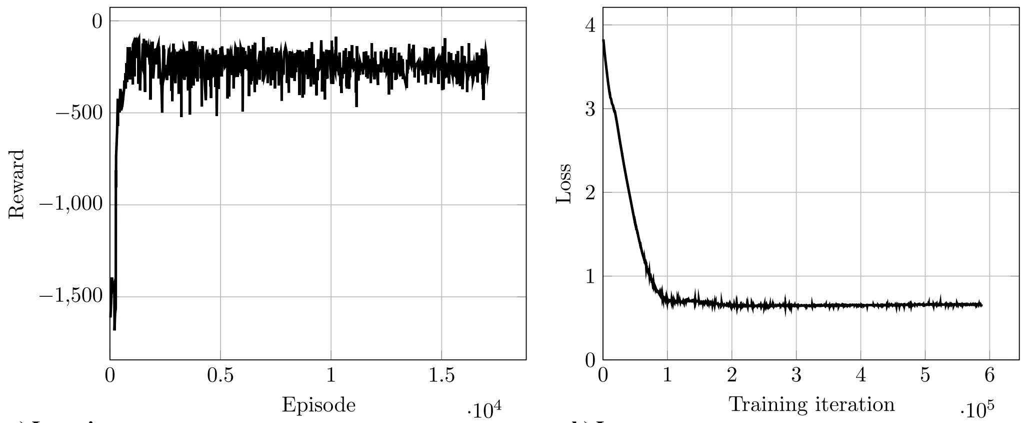


Fig. 10 Examples of learned chaser trajectories with a rotating target.



a) Learning curve **b) Loss**

Fig. 11 Training performance for chaser pose tracking and docking while avoiding an obstacle.

500 Hz by a pulse-width-modulation scheme using solenoid valves. Pressurized air for the thrusters and the air bearing flotation system is stored onboard in a single air cylinder at 31 MPa (4500 psi). The structure consists of an aluminum frame with four corner rods on which three modular decks are stacked. To protect the internal components, the structure is covered with semitransparent acrylic panels. Figure 13a shows the SPOT laboratory facility, and Fig. 13b shows two SPOT platforms in a proximity operations configuration.

The motion of both platforms is measured in real time through four active light-emitting diodes (LEDs) on each platform, which are tracked by an eight-camera PhaseSpace© motion capture system. This provides highly accurate ground-truth position and attitude data. All motion capture cameras are connected to a PhaseSpace server, which is connected to a ground station computer. The ground station computer wirelessly communicates ground truth information to the onboard computers of the platforms, which consist of Raspberry Pi 3s running the Raspbian Linux operating system. Based on the position and attitude data the platforms wirelessly receive, they can perform feedback control by calculating the required thrust to maneuver autonomously and actuating the appropriate solenoid valves to realize this motion. The ground station computer also receives real-time telemetry data (i.e., any signals of interest, as specified by the user) from all onboard computers, for post-experiment analysis purposes.

A MATLAB/Simulink® numerical simulator that recreates the dynamics and emulates the different onboard sensors and actuators is first used to design and test the upcoming experiment. Once the performance in simulations is satisfactory, the control software is converted into C/C++ using Embedded Coder®, compiled, and then executed on the Raspberry Pi-3 computers of the platforms.

An NVIDIA Jetson TX2 Module is used to run the trained deep guidance policy neural network in real time. It accepts the current system state and returns a guidance velocity signal to the Raspberry Pi-3 that executes a control law to track the commanded velocity.

B. Setup

The simulations presented in Sec. V were chosen such that they are replicable experimentally. In other words, all three pose tracking and docking tasks with randomized initial conditions are attempted in experiment. The final parameters θ of the trained deep guidance policies are exported for use on the chaser SPOT platform. The value

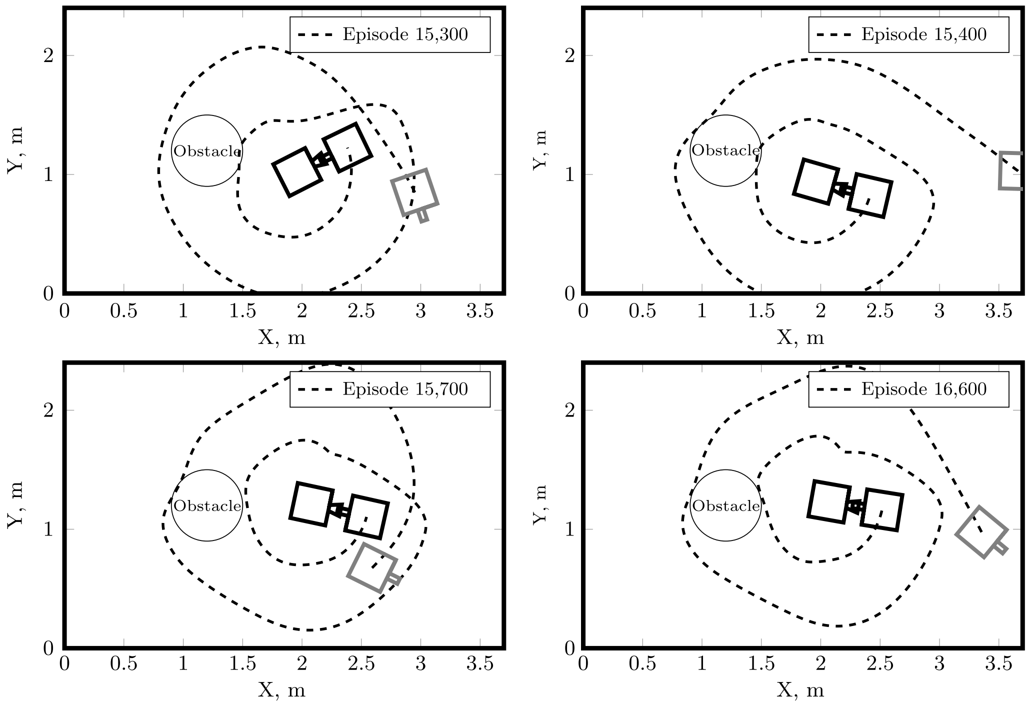
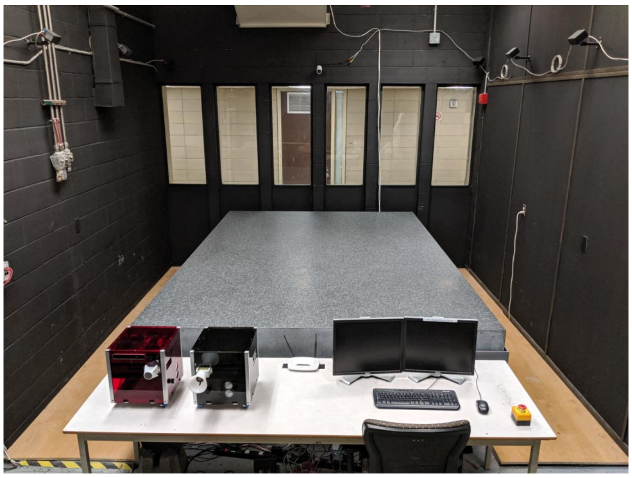
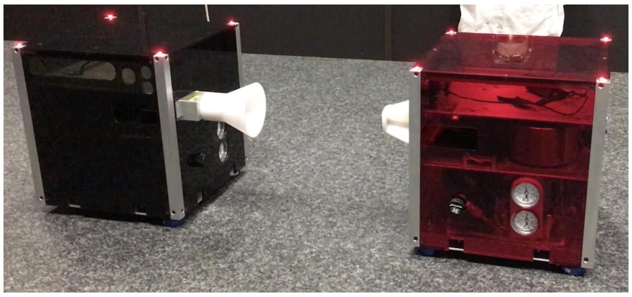


Fig. 12 Examples of learned chaser trajectories while avoiding an obstacle.



a) An overview of the SPOT facility



b) SPOT platforms

Fig. 13 Spacecraft Proximity Operations Testbed.

network is only used during training and is not exported along with the policy network. Initial conditions are similar to those used previously in simulation.

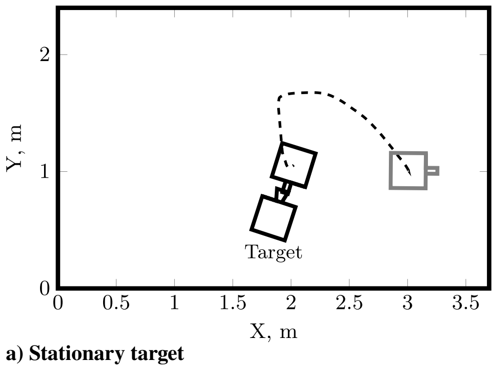
The platforms remain in contact with the table until a strong lock has been acquired on the LEDs by the motion capture system. Following this, the platforms begin to float and maneuver to the desired initial conditions. Then, the target remains stationary or begins rotating while the chaser platform uses the deep guidance policy trained in simulation to guide itself toward the hold point and finally the docking point on the target.

It should be noted that a significant number of discrepancies exist between the simulated environment the policy was trained within and the experimental facility. The simulated environment did not account for the discrete thrusters and their limitations, the control thrust allocation strategy, signal noise, system delays, friction, air resistance, center of mass offsets, thruster plume interaction, and table slope. In addition, the spacecraft mass used to evaluate the training was 10 kg, whereas the experimental spacecraft platforms have a mass of 16.9 kg. Significant discrepancies exist between the simulated training environment and the experimental environment, which is an excellent test for the simulation-to-reality capabilities of the proposed deep guidance technique. Because of facility size limitations, the hold point was reduced from 1.0 to 0.9 m offset from the front face of the target.

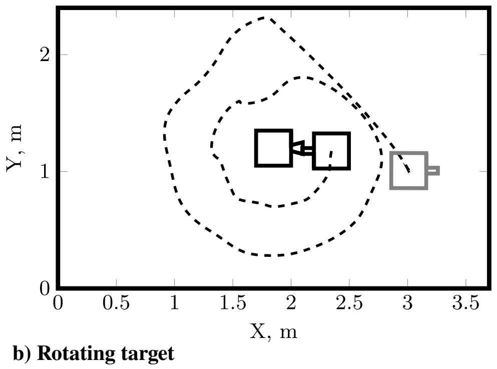
C. Results

A trajectory of the experiment with a stationary target is shown in Fig. 14a. It shows the the deep guidance successfully outputs velocity commands that bring the chaser to the hold point, and then additional velocity commands to move toward the target docking port. A trajectory of the experiment with a rotating target is shown in Fig. 14b, and a trajectory of the experiment with an obstacle and a rotating target is shown in Fig. 14c. The learned deep guidance technique successfully commands an appropriate velocity signal for the chaser to complete the tasks.

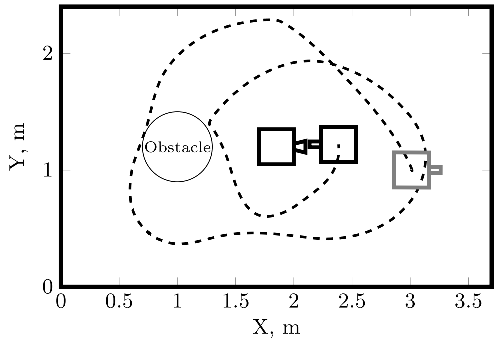
The deep guidance technique was successfully trained exclusively in simulation and deployed to an experimental facility, and achieved similar performance to that during training. Combining the neural-network guidance with conventional control allowed the trained system to handle unmodeled effects present in the experiment. The proposed technique successfully demonstrates the deep guidance technique as a viable solution to the simulation-to-reality problem present in deep reinforcement learning. A video of the simulated and experimental results can be found in supplemental video S1 or online at https://youtu.be/n7 K6aC5v0aY.



a) Stationary target



b) Rotating target



c) Rotating target with obstacle avoidance

Fig. 14 Experimental trajectories of deep guidance in SPOT of Carleton University.

VII. Conclusions

This paper introduced deep reinforcement learning to the guidance problem for spacecraft robotics. Through training a guidance policy to accomplish a goal, complex behaviors can be learned rather than handcrafted. The simulation-to-reality gap dictates that policies trained in simulation often do not transfer well to reality. To avoid this, and to avoid additional training once deployed to a physical robot, the authors restrict the policy to output a guidance signal, which a conventional controller is tasked with tracking. Conventional control can handle the modeling errors that typically plague reinforcement learning. This paper tests this learned guidance technique, which the authors call deep guidance, on a simple problem, that is, spacecraft pose tracking and docking. Numerical simulations show that proximity operation tasks can be successfully learned using the deep guidance technique. The trained policies are then deployed to three experiments, with comparable results to those in simulation, even though the simulated environment did not model all effects present in the experimental facility. Future work will further explore the generality of the technique and its use on more-difficult problems.

Acknowledgment

This research was financially supported in part by the Natural Sciences and Engineering Research Council of Canada under the Postgraduate Scholarship—Doctoral PGSD3-503919-2017 award.

References

[1] Flores-Abad, A., Ma, O., Pham, K., and Ulrich, S., "A Review of Space Robotics Technologies for On-Orbit Servicing," *Progress in Aerospace Sciences*, Vol. 68, July 2014, pp. 1–26.
https://doi.org/10.1016/j.paerosci.2014.03.002

[2] Aghili, F., "A Prediction and Motion-Planning Scheme for Visually Guided Robotic Capturing of Free-Floating Tumbling Objects with Uncertain Dynamics," *IEEE Transactions on Robotics*, Vol. 28, No. 3, 2012, pp. 634–649.
https://doi.org/10.1109/TRO.2011.2179581

[3] Wilde, M., Ciarcià, M., Grompone, A., and Romano, M., "Experimental Characterization of Inverse Dynamics Guidance in Docking with a Rotating Target," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 6, 2016, pp. 1173–1187.
https://doi.org/10.2514/1.G001631

[4] Ma, Z., Ma, O., and Shashikanth, B. N., "Optimal Approach to and Alignment with a Rotating Rigid Body for Capture," *Journal of the Astronautical Sciences*, Vol. 55, No. 4, 2007, pp. 407–419.
https://doi.org/10.1007/BF03256532

[5] Dong, H., Hu, Q., and Akella, M. R., "Dual-Quaternion-Based Spacecraft Autonomous Rendezvous and Docking Under Six-Degree-of-Freedom Motion Constraints," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, 2018, pp. 1150–1162.
https://doi.org/10.2514/1.G003094

[6] Filipe, N., and Tsiotras, P., "Adaptive Position and Attitude Tracking Controller for Satellite Proximity Operations Using Dual Quaternions," *Journal of Guidance, Control, and Dynamics*, Vol. 38, No. 4, 2015, pp. 566–577.
https://doi.org/10.2514/1.G000054

[7] Gui, H., and Vukovich, G., "Finite-Time Output-Feedback Position and Attitude Tracking of a Rigid Body," *Automatica*, Vol. 74, Dec. 2016, pp. 270–278.
https://doi.org/10.1016/j.automatica.2016.08.003

[8] Pothen, A. A., and Ulrich, S., "Close-Range Rendezvous with a Moving Target Spacecraft Using Udwadia-Kalaba Equation," *American Control Conference*, IEEE Publ., Piscataway, NJ, 2019, pp. 3267–3272.
https://doi.org/10.23919/ACC.2019.8815115

[9] Pothen, A. A., and Ulrich, S., "Pose Tracking Control for Spacecraft Proximity Operations Using the Udwadia-Kalaba Framework," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2020-1598, Jan. 2020.
https://doi.org/10.2514/6.2020-1598

[10] Hough, J., and Ulrich, S., "Lyapunov Vector Fields for Thrust-Limited Spacecraft Docking with an Elliptically-Orbiting Uncooperative Tumbling Target," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2020-2078, Jan. 2020.
https://doi.org/10.2514/6.2020-2078

[11] Hornik, K., Stinchcombe, M., and White, H., "Multilayer Feedforward Networks Are Universal Approximator," *Neural Networks*, Vol. 2, No. 5, 1989, pp. 359–366.
https://doi.org/10.1016/0893-6080(89)90020-8

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-Level Control Through Deep Reinforcement Learning," *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533.
https://doi.org/10.1038/nature14236

[13] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M., "The Arcade Learning Environment: An Evaluation Platform for General Agents,"

Journal of Artificial Intelligence Research, Vol. 47, June 2013, pp. 253–279.
https://doi.org/10.1613/jair.3912

[14] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D., "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, Vol. 529, No. 7587, 2016, pp. 484–489.
https://doi.org/10.1038/nature16961

[15] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D., "Mastering the Game of Go Without Human Knowledge," *Nature*, Vol. 550, No. 7676, 2017, pp. 354–359.
https://doi.org/10.1038/nature24270

[16] Kober, J., Bagnell, J. A., and Peters, J., "Reinforcement Learning in Robotics: A Survey," *International Journal of Robotics Research*, Vol. 32, No. 11, 2013, pp. 1238–1274.
https://doi.org/10.1177/0278364913495721

[17] Tai, L., Paolo, G., and Liu, M., "Virtual-to-Real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation," *IEEE International Conference on Intelligent Robots and Systems*, IEEE Publ., Piscataway, NJ, 2017, pp. 31–36.
https://doi.org/10.1109/IROS.2017.8202134

[18] Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., and Corke, P., "The Limits and Potentials of Deep Learning for Robotics," *International Journal of Robotics Research*, Vol. 37, Nos. 4–5, 2018, pp. 405–420.
https://doi.org/10.1177/0278364918770733

[19] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V., "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping," *IEEE International Conference on Robotics and Automation*, IEEE Publ., Piscataway, NJ, 2018, pp. 4243–4250.
https://doi.org/10.1109/ICRA.2018.8460875

[20] Andrychowicz, O. A. M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W., "Learning Dexterous In-Hand Manipulation," *International Journal of Robotics Research*, Vol. 39, No. 1, 2020, pp. 3–20.
https://doi.org/10.1177/0278364919887447

[21] Loquercio, A., Kaufmann, E., Ranftl, R., Dosovitskiy, A., Koltun, V., and Scaramuzza, D., "Deep Drone Racing: From Simulation to Reality with Domain Randomization," *IEEE Transactions on Robotics*, Vol. 36, No. 1, 2020, pp. 1–14.
https://doi.org/10.1109/TRO.2019.2942989

[22] Cutler, M., and How, J. P., "Autonomous Drifting Using Simulation-Aided Reinforcement Learning," *IEEE International Conference on Robotics and Automation*, IEEE Publ., Piscataway, NJ, 2016, pp. 5442–5448.
https://doi.org/10.1109/ICRA.2016.7487756

[23] Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M., "Control of a Quadrotor with Reinforcement Learning," *IEEE Robotics and Automation Letters*, Vol. 2, No. 4, 2017, pp. 2096–2103.
https://doi.org/10.1109/LRA.2017.2720851

[24] Julian, K. D., and Kochenderfer, M. J., "Distributed Wildfire Surveillance with Autonomous Aircraft Using Deep Reinforcement Learning," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 8, 2019, pp. 1768–1778.
https://doi.org/10.2514/1.G004106

[25] Chan, D. M., and Agha-Mohammadi, A. A., "Autonomous Imaging and Mapping of Small Bodies Using Deep Reinforcement Learning," *IEEE Aerospace Conference*, IEEE Publ., Piscataway, NJ, 2019.
https://doi.org/10.1109/AERO.2019.8742147

[26] Willis, S., Izzo, D., and Hennes, D., "Reinforcement Learning for Spacecraft Maneuvering Near Small Bodies," *AAS/AIAA Space Flight Mechanics Meeting*, AAS Paper 16-277, Feb. 2016, pp. 1351–1368.

[27] Lafarge, N. B., Miller, D., Howell, K. C., and Linares, R., "Guidance for Closed-Loop Transfers Using Reinforcement Learning with Application to Libration Point Orbits," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2020-0458, Jan. 2020.
https://doi.org/10.2514/6.2020-0458

[28] Scorsoglio, A., Furfaro, R., Linares, R., and Gaudet, B., "Image-Based Deep Reinforcement Learning for Autonomous Lunar Landing," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2020-1910, Jan. 2020.
https://doi.org/10.2514/6.2020-1910

[29] Gaudet, B., and Furfaro, R., "Adaptive Pinpoint and Fuel Efficient Mars Landing Using Reinforcement Learning," *IEEE/CAA Journal of Automatica Sinica*, Vol. 1, No. 4, 2014, pp. 397–411.
https://doi.org/10.1109/JAS.2014.7004667

[30] Furfaro, R., Simo, J., Gaudet, B., and Wibben, D. R., "Neural-Based Trajectory Shaping Approach for Terminal Planetary Pinpoint Guidance," *AAS/AIAA Astrodynamics Specialist Conference*, AAS Paper 13-875, Aug. 2013.

[31] Sánchez-Sánchez, C., and Izzo, D., "Real-Time Optimal Control via Deep Neural Networks: Study on Landing Problems," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, 2018, pp. 1122–1135.
https://doi.org/10.2514/1.G002357

[32] Ulrich, S., Saenz-Otero, A., and Barkana, I., "Passivity-Based Adaptive Control of Robotic Spacecraft for Proximity Operations Under Uncertainties," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 6, 2016, pp. 1444–1453.
https://doi.org/10.2514/1.G001491

[33] Jafarnejadsani, H., Sun, D., Lee, H., and Hovakimyan, N., "Optimized L1 Adaptive Controller for Trajectory Tracking of an Indoor Quadrotor," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 6, 2017, pp. 1415–1427.
https://doi.org/10.2514/1.G000566

[34] Huang, P., Wang, D., Meng, Z., Zhang, F., and Guo, J., "Adaptive Postcapture Backstepping Control for Tumbling Tethered Space Robot-Target Combination," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 1, 2016, pp. 150–156.
https://doi.org/10.2514/1.G001309

[35] Harris, A., Teil, T., and Schaub, H., "Spacecraft Decision-Making Autonomy Using Deep Reinforcement Learning," *AAS/AIAA Space Flight Mechanics Meeting*, AAS Paper 19-447, Jan. 2019.

[36] Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Dhruva, T. B., Muldal, A., Heess, N., and Lillicrap, T., "Distributed Distributional Deterministic Policy Gradients," *International Conference on Learning Representations*, Vancouver, Canada, 2018.

[37] Bellemare, M. G., Dabney, W., and Munos, R., "A Distributional Perspective on Reinforcement Learning," *International Conference on Machine Learning*, PMLR, Sydney, Australia, 2017, pp. 449–458.

[38] Mnih, V., Badia, A., Mirza, M., Graves, A., and Lillicrap, T., "Asynchronous Methods for Deep Reinforcement Learning," *International Conference on Machine Learning*, PMLR, New York, 2016, pp. 1928–1937.

[39] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, Cambridge, MA, 1998, p. 148.

[40] Zappulla, R., Park, H., Virgili-Llop, J., and Romano, M., "Real-Time Autonomous Spacecraft Proximity Maneuvers and Docking Using an Adaptive Artificial Potential Field Approach," *IEEE Transactions on Control Systems Technology*, Vol. 27, No. 6, 2019, pp. 2598–2605.
https://doi.org/10.1109/TCST.2018.2866963

[41] Ciarcià, M., Grompone, A., and Romano, M., "A Near-Optimal Guidance for Cooperative Docking Maneuvers," *Acta Astronautica*, Vol. 102, Sept. 2014, pp. 367–377.
https://doi.org/10.1016/j.actaastro.2014.01.002

[42] Mammarella, M., Capello, E., Park, H., Guglieri, G., and Romano, M., "Tube-Based Robust Model Predictive Control for Spacecraft Proximity Operations in the Presence of Persistent Disturbance," *Aerospace Science and Technology*, Vol. 77, June 2018, pp. 585–594.
https://doi.org/10.1016/j.ast.2018.04.009

[43] Saulnier, K., Pérez, D., Huang, R. C., Gallardo, D., Tilton, G., and Bevilacqua, R., "A Six-Degree-of-Freedom Hardware-in-the-Loop Simulator for Small Spacecraft," *Acta Astronautica*, Vol. 105, No. 2, 2014, pp. 444–462.
https://doi.org/10.1016/j.actaastro.2014.10.027

[44] Oliphant, T. E., "Python for Scientific Computing," *Computing in Science & Engineering*, Vol. 9, No. 3, 2007, pp. 10–20.
https://doi.org/10.1109/MCSE.2007.58

[45] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous Control with Deep Reinforcement Learning," *International Conference on Learning Representations*, San Juan, Puerto Rico, 2016.

[46] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, San Diego, CA, 2015.

I. I. Hussein
Associate Editor