Acta Astronautica 186 (2021) 1–23

Contents lists available at ScienceDirect

Acta Astronautica

journal homepage: www.elsevier.com/locate/actaastro

Research paper

# Autonomous closed-loop guidance using reinforcement learning in a low-thrust, multi-body dynamical environment☆

Nicholas B. LaFarge [a,*], Daniel Miller [b], Kathleen C. Howell [a], Richard Linares [b]

[a] School of Aeronautics and Astronautics, Purdue University, West Lafayette, IN, 47907, USA
[b] Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA

ARTICLE INFO

*Keywords:*
Spacecraft autonomy
Closed-loop guidance
Reinforcement learning
Low-thrust
Cislunar space
Neural network control

ABSTRACT

Onboard autonomy is an essential component in enabling increasingly complex missions into deep space. In nonlinear dynamical environments, computationally efficient guidance strategies are challenging. Many traditional approaches rely on either simplifying assumptions in the dynamical model or on abundant computational resources. This research effort employs reinforcement learning, a subset of machine learning, to produce a 'lightweight' closed-loop controller that is potentially suitable for onboard low-thrust guidance in challenging dynamical regions of space. The results demonstrate the controller's ability to directly guide a spacecraft despite large initial deviations and to augment a traditional targeting guidance approach. The proposed controller functions without direct knowledge of the dynamical model; direct interaction with the nonlinear equations of motion creates a flexible learning scheme that is not limited to a single force model, mission scenario, or spacecraft. The learning process leverages high-performance computing to train a closed-loop neural network controller. This controller may be employed onboard to autonomously generate low-thrust control profiles in real-time without imposing a heavy workload on a flight computer. Control feasibility is demonstrated through sample transfers between Lyapunov orbits in the Earth–Moon system. The sample low-thrust controller exhibits remarkable robustness to perturbations and generalizes effectively to nearby motion. Finally, the flexibility of the learning framework is demonstrated across a range of mission scenarios and low-thrust engine types.

## 1. Introduction

Advancements in onboard autonomy are enabling new opportunities for establishing a sustained human and robotic presence in deep space. In complex multi-body dynamical environments, such as in the Earth–Moon neighborhood, onboard applications for low-thrust spacecraft are particularly challenging. This investigation demonstrates Reinforcement Learning (RL), a subset of Machine Learning (ML), to be an effective approach for automated closed-loop guidance in these challenging regions of space. Onboard autonomous guidance requires a computationally efficient approach that addresses nonlinearity in the dynamical model and offers flexibility as requirements change inflight. In satisfying these criteria, RL provides a model-agnostic approach for training a neural network controller that is applicable to multiple problems, and potentially suitable for onboard use.

Future spacecraft must be capable of accessing nonlinear regions of space given various propulsive capabilities. Several upcoming mission concepts include trajectories that involve complex dynamical structures that only exist in force models that incorporate the gravity of multiple celestial bodies simultaneously. In particular, with the proposed Gateway and Artemis projects, NASA aims to establish a permanent foothold in cislunar space. To ensure crew safety, future spacecraft, such as Orion, require the capability to perform all GNC functionality onboard [1]. Furthermore, robotic missions venturing far from the Earth into multi-body regimes, such as Europa Clipper, similarly benefit from increased autonomous capability [2]. While recent progress has addressed these dynamical challenges within the context of the Circular Restricted Three-Body Problem (CR3BP), an additional complicating factor is the planned inclusion of Solar Electric Propulsion (SEP) options that prohibit instantaneous maneuvers in favor of gradual velocity and energy changes over longer time intervals. Upcoming low-thrust missions include nanosatellites such as Lunar IceCube [3] and LunaH-Map [4], as well as larger missions such as Psyche [5] and the Lunar

☆ Some content presented previously as: LaFarge, N.B., Miller, D., Howell, K. C., Linares, R. "Guidance for closed-loop transfers using reinforcement learning with application to libration point orbits," AIAA SciTech Forum, Orlando, Florida, January 6 – 10, 2020.

* Corresponding author.
*E-mail addresses:* nlafarge@purdue.edu (N.B. LaFarge), dmmiller@mit.edu (D. Miller), howell@purdue.edu (K.C. Howell), linaresr@mit.edu (R. Linares).

https://doi.org/10.1016/j.actaastro.2021.05.014
Received 4 December 2020; Received in revised form 11 February 2021; Accepted 7 May 2021
Available online 11 May 2021

Gateway [6]. Overcoming sensitive dynamics is challenging given limited control authority. Furthermore, corresponding solutions often rely on the underlying assumptions in the force model and are not easily extendable to higher-fidelity domains. Using RL as a model-agnostic guidance approach is more adaptable to different domains and offers direct interaction with complex dynamical models.

Traditionally, guidance techniques involve communicating with ground-based control stations. However, susceptibility to communication failure, time delays, limitations in data transmission, sensor tasking complexity, and operations cost all motivate moving Guidance, Navigation and Control (GNC) functionality to the flight computer. While many recent advancements in trajectory design exploit improvements in computer hardware, relatively few are practical for autonomous onboard implementation given the limited computational resources. Typically, in the pre-flight trajectory design process, the goal is the construction of an optimal trajectory and a control history that meets mission criteria for propellant usage and time of flight. In this pre-flight context, numerous strategies exist for low-thrust optimal guidance including global optimization techniques [7] and nonlinear programming [8]. However, within the context of onboard software, flight computer limitations suggest that feasibility is more important than optimality [9]. In the onboard environment, the capability to rapidly re-compute a reference path and a feasible control history in-flight is critical for autonomous guidance. While some studies consider applying optimization methods onboard, these efforts involve simpler dynamics that do not include a third body [10]. By approaching onboard guidance from a machine learning perspective, a closed-loop neural network controller offers the ability to quickly and autonomously compute a control history for a spacecraft with basic linear algebra operations and without iteration. In addition, RL separates the computationally expensive pre-flight learning from the resulting onboard controller and, thus, leverages modern computer hardware while still producing a controller sufficiently 'lightweight' for use in flight.

Spacecraft trajectory targeting and optimization methods, in general, require suitable startup solutions. In multi-body problems, trajectory designers often generate low-cost initial guesses for transfers by leveraging dynamical systems theory and invariant manifolds [11,12]. Dynamical systems-based approaches have been useful in many previous applications and, when combined with differential corrections and/or optimization techniques, yield optimal solutions for many applications. However, this approach is computationally intensive and often requires human-in-the-loop interactions. Alternatively, global optimization techniques, such as basin-hopping and evolutionary algorithms, reduce the need for accurate startup solutions [7], but the corresponding computational complexity renders them infeasible for onboard use. Without a trajectory designer in-the-loop, onboard startup arcs are typically generated in one of two ways. The most common approach is to store a database of optimal solutions on the flight computer, and adjust the solutions as necessary during flight [13]. Alternatively, a recent approach is the generation of infeasible startup arcs based on two-body approximations, and correct discontinuities with onboard targeting [9]. In multi-body regimes, conic approximations are frequently insufficient, and limit the onboard guidance capability [12].

Neural networks offer several attractive features for enabling autonomous guidance. First, a neural network is capable of directly mapping state estimates to thrust commands that satisfy mission constraints. Neural network guidance performance is demonstrated in studies such as planetary landing [14], asteroid proximity operations [15], and missed thrust problems [16]. Alternatively, a neural network can augment, rather than replace, existing guidance schemes. For onboard targeting, it is challenging to quickly generate a suitable initial guess for a control history. This investigation demonstrates a neural network's ability to produce startup arcs for low-thrust targeting. In combining neural network and targeting paradigms, the resulting hybrid guidance framework simultaneously benefits from the speed of neural networks, and the robustness of targeting to ensure all mission criteria are met.

Researchers have recently applied various types of machine learning techniques to onboard autonomy problems, with studies ranging from preliminary investigations to demonstrated onboard capability. For example, a basic machine learning algorithm is implemented onboard Mars rovers to aid in identifying geologic features of interest. The Autonomous Exploration for Gathering Increased Science (AEGIS) algorithm enables autonomous ChemCam target selection onboard Spirit, Opportunity, and Curiosity [17]. On Curiosity's main flight computer, the pointing refinement process requires between 94 and 96 s [18] – substantially less than the time necessary to send images to Earth and wait for scientists to manually select features. Upcoming onboard machine learning applications include autonomous onboard robotic capabilities for the Perseverance rover [19,20], and anomaly detection for Europa Clipper [2]. Machine learning algorithms offer the potential for significant contributions to future autonomous missions.

Beyond planetary robotics, several investigations apply supervised learning techniques toward astrodynamics problems. In trajectory design, regression tasks are typically the most productive. Dachwald applied a shallow Neural Network (NN) to low-thrust trajectory optimization as early as 2004 [21]. More recent investigations involve heteroclinic transfer identification [22], low-thrust trajectory corrections [23], and missed thrust analysis [16]. Supervised learning techniques can produce desirable outcomes but carry several notable drawbacks. First, such approaches assume knowledge of the decision-making process. By selecting desired outcomes, the user assumes *a-priori* knowledge of the basis for decision making. This assumption implies that (1) the user-generated data accurately reflects the desired outcomes; and (2) techniques are available to solve the current problem and to generate the data. In regimes where such knowledge is absent, supervised learning techniques are not applicable.

In recent years, RL has proven beneficial in achieving state-of-the-art performance in historically challenging domains despite significant environmental uncertainty [24,25]. Reinforcement learning-enabled onboard guidance is broadly categorized based on the phase of flight. Notably productive areas of RL research are landing problems [14,26] and small body operations [15,27]. Other investigations include rendezvous [28], exo-atmospheric interception [29], stationkeeping [30], and detection avoidance [31]. Studies that involve low-thrust spacecraft in a multi-body dynamical regime include transfer design using $Q$-Learning [32] and Proximal Policy Optimization (PPO) [33], as well as orbit approach guidance [34].

The present investigation seeks to demonstrate a novel framework for closed-loop guidance within a multi-body regime. Without direct knowledge of the complex environmental dynamics, reinforcement learning is leveraged to produce a neural network controller that successfully guides a spacecraft to its destination given large initial deviations. Once trained, the resulting neural network offers multiple benefits for potential onboard use. For instance, the guidance scheme is sufficiently 'lightweight' for use on a flight computer, is robust to changing reference trajectories, and is adaptable to multiple problems. Furthermore, this investigation demonstrates the controller's ability to directly guide a spacecraft, and to augment a traditional guidance approach by providing an initial guess to a direct multiple shooting targeting algorithm.

The problem formulation for the dynamical model and the differential corrections framework is presented in Section 2. Section 3 offers an overview of RL and PPO, and details the learning environment and the implementation scheme employed in this investigation. Experimental results in Section 4 demonstrate the proposed guidance framework in a sample application involving transfers between libration point orbits in the Earth–Moon system. Section 5 details the learning framework's performance across a range of thrust levels, reference trajectory geometries, and energy levels. Finally, algorithm limitations are discussed in Section 6.

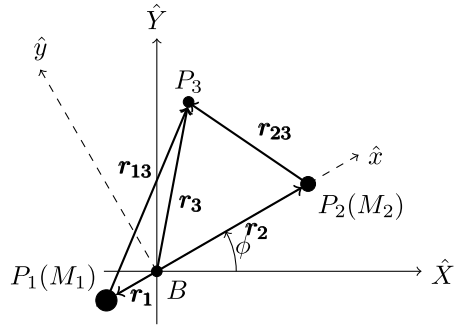



**Fig. 1.** Planar CR3BP vector definitions. The rotating frame $(\hat{x}, \hat{y})$ is oriented with respect an inertial reference frame $(\hat{X}, \hat{Y})$ by an angle $\phi$.
*Source:* Modified from Cox [35].

## 2. Problem formulation

The three-body problem serves as a suitable dynamical model that is representative of observed natural motion in cislunar space. While the proposed guidance framework does not depend on any particular model, the planar Circular Restricted Three-Body Problem (CR3BP) serves as a useful environment for preliminary evaluation because it both represents a challenging region of space that is relevant to upcoming missions while sufficiently low-fidelity for initial analysis of the guidance scheme. Additionally, low-thrust propulsion is included to demonstrate algorithmic performance despite limited control authority and pronounced nonlinearities. While the proposed guidance scheme directly supplies a control history, an alternative concept of operations leverages the trained neural network to produce an initial guess for other numerical methods, such as targeting or optimization schemes. A sample direct multiple shooting algorithm is included in this analysis to demonstrate the added value of the neural network to the onboard targeting capability.

### 2.1. Dynamical model

The CR3BP is a model for the motion of an infinitesimal mass moving under the influence of two celestial bodies. In this model, two spherically symmetric gravitational bodies, $P_1$ and $P_2$ form the primary system as they move in circular orbits about their common barycenter, $B$; $P_3$ moves freely with respect to the barycenter, as depicted in Fig. 1. The relative size of the primaries is represented by the mass ratio $\mu = M_2/(M_1 + M_2)$, with $M_1$ assumed to be the larger of the two bodies. Furthermore, this model assumes that the mass of $P_3$ is infinitesimal compared to the masses of the primary bodies and, thus, does not influence the motion of the primary system. The position and velocity of $P_3$, denoted $\boldsymbol{r_3}$, $\boldsymbol{v_3}$, respectively, comprise the vector $\boldsymbol{\rho}^{\text{spatial}} = [x \;\; y \;\; z \;\; \dot{x} \;\; \dot{y} \;\; \dot{z}]^T$. The vector components are propagated with respect to the system barycenter, $B$, in a rotating reference frame, denoted by dashed lines in Fig. 1.

Many mission architectures benefit from the inclusion of low-thrust electric propulsion. In contrast to traditional chemical engines, electric propulsive engines are much more efficient, but deliver energy changes over longer time intervals. Low-thrust engines using Solar Electric Propulsion (SEP) include ion thrusters, which are powered through solar panels on the spacecraft. Currently, ion thrusters are successfully employed, for example, on Deep Space 1 [36] and Dawn [37], as well as other missions. Building on this progress, upcoming low-thrust missions include Psyche [5] and the Lunar Gateway [6]. This investigation assumes that $P_3$ is a spacecraft with a Constant Specific Impulse (CSI) low-thrust engine. The additional propulsion force augments the natural CR3BP equations of motion with low thrust terms,

$$\ddot{x} - 2\dot{y} - x = -\frac{(1-\mu)(x+\mu)}{r_{13}^3} - \frac{\mu(x-1+\mu)}{r_{23}^3} + a_{\text{lt}}u_x \tag{1}$$

$$\ddot{y} + 2\dot{x} - y = -\frac{(1-\mu)y}{r_{13}^3} - \frac{\mu y}{r_{23}^3} + a_{\text{lt}}u_y \tag{2}$$

$$\ddot{z} = -\frac{(1-\mu)z}{r_{13}^3} - \frac{\mu z}{r_{23}^3} + a_{\text{lt}}u_z \tag{3}$$

$$\dot{m} = 0 + \frac{-fl^*}{I_{\text{sp}}g_0t^*} \tag{4}$$

where red denotes the additional terms introduced by the low-thrust force, $t^*$ and $l^*$ are the system characteristic time and length, respectively, and $g_0 = 9.80665 \times 10^{-3}$ km/s. The distances between the first and second primary body are defined as $r_{13} = \sqrt{(x+\mu)^2 + y^2 + z^2}$ and $r_{23} = \sqrt{(x-1+\mu)^2 + y^2 + z^2}$, respectively. Motion in the CR3BP is nonlinear and describes notably sensitive dynamical structures, thus, the proposed guidance strategy exploits the impact of the low-thrust terms to achieve desired behavior. As detailed by Cox et al. [38], thrust direction is defined by the low-thrust acceleration vector,

$$\boldsymbol{a}_{\text{lt}} = \frac{f}{m}\hat{u} = (a_{\text{lt}}u_x)\hat{x} + (a_{\text{lt}}u_y)\hat{y} + (a_{\text{lt}}u_z)\hat{z} \tag{5}$$

where $f$ is the nondimensional thrust magnitude, $m = M_3/M_{3,0}$ is the nondimensional spacecraft mass, and $M_3$ is the mass of the spacecraft at the beginning of the thrusting segment. The thrust direction is oriented by a unit vector, $\hat{u}$, fixed in the rotating frame. Over any integration segment, thrust is assumed fixed in the CR3BP rotating frame. While a continuously changing inertial thrust direction may not be practical, this investigation involves 20.87 h thrust segments, and the precession of the rotating frame during these time intervals may be addressed when transferring CR3BP solutions into a higher-fidelity model. Furthermore, propulsive capability is inversely related to spacecraft mass and, hence, as propellant is expended, the spacecraft gains more thrust. The nondimensional thrust magnitude is computed as,

$$f = \frac{Ft^*}{l^*M_{3,0}} \tag{6}$$

where $F$ is thrust in kilonewtons and $M_{3,0}$ is the initial mass of the spacecraft in kilograms. This investigation includes a sample spacecraft with maximum propulsive capability of $f_{\max} = 0.04$. A comparison between this sample spacecraft and other previous and planned engine capabilities is summarized in Table 1. The sample spacecraft possesses similar propulsive capability to the planned Psyche spacecraft, which is more than Hayabusa 1, Hayabusa 2, Dawn, and Lunar IceCube, but less than Deep Space 1. As a nondimensional quantity, this thrust level represents any spacecraft with the same thrust-to-mass ratio. For example, $f_{\max} = 0.04$ models an 11.46 kg spacecraft with a BIT-3 CubeSat engine [39] (planned for use on Lunar IceCube and LunaH-Map) as well as an 850.2 kg spacecraft with an NSTAR engine [36] (used on Deep Space 1 and Dawn).

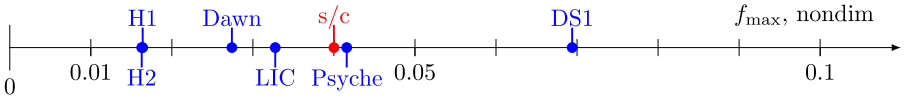
To provide a simpler environment for assessing the learning framework, subsequent examples assume all motion occurs in the $x$-$y$ plane. In particular, the planar assumption implies that the spatial components $(z, \dot{z}, u_z)$, are all zero. Hence, the vector $\boldsymbol{\rho}^{\text{spatial}}$ becomes $\boldsymbol{\rho} = [x \;\; y \;\; \dot{x} \;\; \dot{y}]^T$, where all the spatial terms are eliminated. Omitting low-thrust terms, the natural CR3BP equations of motion yield a well-known integral, i.e., the Jacobi constant of integration ($C$). This single integral of motion is evaluated as,

$$C = 2\left(\frac{1-\mu}{r_{13}} + \frac{\mu}{r_{23}} + \frac{1}{2}\left(x^2 + y^2\right)\right) - \left(\dot{x}^2 + \dot{y}^2\right) \tag{7}$$

The Jacobi constant is related to orbital energy as observed in the rotating frame, and remains constant throughout any natural propagation. This integral offers key insight into the limiting bounds for possible motion of $P_3$ and supplies a useful scalar metric for deviations relative to a reference trajectory where $C$ is preserved. Without a known analytical solution in the CR3BP, numerical integration methods are leveraged

**Table 1**
Low-thrust capability of various spacecraft, nondimensionalized in the Earth–Moon system.

| Abbrv. | Spacecraft | $f_{max}$, nondim | $M_{3,0}$, kg | $F_{max}$, mN |
|---|---|---|---|---|
| H1 | Hayabusa 1 [40] | $1.640 \cdot 10^{-2}$ | 510 | 22.8 |
| H2 | Hayabusa 2 [41] | $1.628 \cdot 10^{-2}$ | 608.6 | 27.0 |
| LIC | Lunar IceCube [3,39] | $3.276 \cdot 10^{-2}$ | 14 | 1.25 |
| Dawn | Dawn [37] | $2.741 \cdot 10^{-2}$ | 1217.8 | 91.0 |
| DS1 | Deep space 1 [36] | $6.940 \cdot 10^{-2}$ | 486.3 | 92.0 |
| Psyche | Psyche [5,42] | $4.158 \cdot 10^{-2}$ | 2464 | 279.3 |
| s/c | Sample spacecraft | $4 \cdot 10^{-2}$ | n/a | n/a |




to produce a time history stemming from an initial value problem. For implementation, it is useful to express the equations of motion, in Eqs. (1)–(4), as seven coupled first-order equations that are then solved using numerical integration techniques. For this investigation, a Runge–Kutta–Fehlberg 78 integration scheme is employed.

### 2.2. Differential corrections

To demonstrate the augmentation of existing numerical guidance strategies with the proposed neural network controller, a differential corrections scheme, or *targeter*, is employed. Targeting algorithms encompass a broad range of iterative approaches that seek to satisfy a set of scalar constraints by varying parameters. Various targeting approaches are suitable for different aspects of the low-thrust CR3BP [23,43]. This investigation employs a *multiple shooting* strategy to accommodate numerous types of arcs and control variables across a trajectory. This targeter serves as a stand-in for any iterative approach that leverages a startup solution. The targeting scheme here is not intended to satisfy the strict requirements of flight software, but rather to illustrate an RL-generated controller and its incorporation into the initial guess generation process for existing corrections and optimization schemes. A neural network-enabled targeting guidance framework is an appealing hybrid approach, offering the neural network efficiency alongside targeting robustness to ensuring all mission criteria are satisfied. A similar process may be introduced for optimization techniques, but this possibility is not investigated.

The neural network controller produces a continuously thrusting trajectory that terminates in the vicinity of the destination orbit. The targeter is augmented by introducing three modifications to enhance the state and control history:

1. The terminal state is constrained to be located on the stable manifold associated with the destination orbit. This constraint formulation, detailed by Haapala and Howell [11], allows the spacecraft to asymptotically arrive in the desired orbit, and offers flexibility by varying the arrival location along the stable manifold.
2. The targeter is employed to simplify the resulting solution. The RL-generated trajectory contains numerous discrete thrusting segments that are simplified by combining sequential patch points. Averaging the control variables across multiple nodes, a new initial guess is produced, and the multiple shooting process yields an updated feasible solution containing fewer discrete segments.
3. Coasting segments are introduced. When thrust is not necessary, the proposed neural network controller suggests segments with nearly zero thrust magnitude. By applying a thrust threshold, the targeting algorithm allows arcs with a thrust level below a user-defined magnitude to be transformed into coasting segments. A similar thresholding scheme is employed by Das [43]. A further benefit of applying a threshold is a reflection of the fact that many engines are physically unable to produce thrust values below a certain magnitude. For example, Hayabusa 1 and 2 are only capable of maneuvers above 58% and 70% of maximum thrust, respectively [41].

By applying each targeting step sequentially, the targeter transforms the neural network-produced initial guess into a low-thrust solution that contains a single thrust segment with fixed magnitude and direction in the rotating frame.

## 3. Guidance framework design

### 3.1. Reinforcement learning overview

Reinforcement Learning (RL) is a branch of machine learning that encompasses a broad range of goal-oriented algorithms that 'learn' to perform tasks by means of trial-and-error. Current state-of-the-art RL approaches employ modern advancements in neural networks to aid in challenging tasks. Policy gradient methods are of particular recent interest due to their demonstrated ability in continuous control tasks. One such algorithm, Proximal Policy Optimization (PPO), is employed in this investigation to train a neural network controller.

#### 3.1.1. Neural networks

A Neural Network (NN) is a class of nonlinear statistical models that are frequently employed in ML classification and regression tasks [44]. The term *neural network* encompasses many different types of statistical models with various levels of complexity. When applied correctly, NNs perform exceedingly well, and are a driving factor in ML advancements over the past decade. While frequently used in supervised learning applications, NNs are also employed extensively in modern RL algorithms due to their demonstrated ability in approximating nonlinear functions. Many traditional tabular RL approaches, such as $Q$-learning, rely on finely discretizing the state and action spaces, quickly becoming impractical as the number of dimensions in the problem increases. Leveraging NNs allows modern algorithms to both access continuous state and action spaces and to easily incorporate additional dimensions.

Evaluating a feedforward neural network consists of straightforward linear algebra operations, with several nonlinear element-wise activation functions. After the input layer, each node is computed as a linear combination of weighted values and a bias, and processed through an activation function to incorporate nonlinearity into the model [44]. The weights signify the impact that particular nodes exert on each node in the next layer. The bias allows the activation function to be shifted left or right for each node. Together, the weights and biases form the set of trainable parameters for the model. In general, these parameters cannot be known *a-priori*, and so no suitable initial guess for their value is possible. Hence, the weights in this investigation are randomly initialized according to a normal distribution of zero mean and the biases are initialized to zero.

Without activation functions, the network is only able to model linear functions and, thus, the selection of the activation function is an important component in neural network performance. Furthermore,

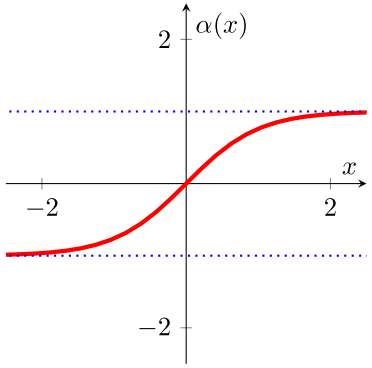



**Fig. 2.** Hyperbolic tangent neural network activation function employed in this investigation.

bounded functions are advantageous since they aid in normalizing the output of each neuron. This investigation employs hyperbolic tangent (tanh) to bound outputs and incorporate nonlinearity. Linear activation is also, at times, advantageous. Within RL, networks that produce a single scalar output value frequently employ a linear activation in the output layer. Together, tanh and linear are the only two activation functions employed in this investigation, with tanh plotted in Fig. 2.

### 3.1.2. Onboard considerations

Neural networks are potentially well-suited for use on a flight computer. While training is computationally complex, the evaluation process is relatively simple; several matrix multiplications and additions, combined with element-wise activation functions, encompass the entire process. Furthermore, flight software often requires algorithms to predictably complete in a given number of CPU cycles. For example, Orion orbit guidance is required to complete in a fixed number of steps, which poses significant difficulty in numerical integration and targeting processes [45]. If a function is predictable, a certain amount of processing time is easily allocated. Complexity is introduced when procedures are unpredictable, such as methodologies that require iteration. While computationally lightweight, neural networks are also deterministic which, together with predictability, renders them well-suited to the flight environment.

Despite the relative simplicity in evaluation, implementing a NN on a flight computer presents additional challenges due to "significant amounts of multiply and accumulate operations" and a "substantial amount of memory to store data" [46]. However, the proposed neural network controller is relatively small, containing 10,623 trainable parameters and a memory footprint of 42 KB (0.04 MB) assuming 32-bit floating point precision. Furthermore, specialized "neuromorphic" flight processors are currently being developed to enable low-power NN evaluations in space [46]. Adoption of neuromorphic hardware into flight systems will render machine learning approaches more accessible and productive, and may enable efficient autonomous control, decision making, and onboard adaptive learning [47].

### 3.1.3. Reinforcement learning fundamentals

Reinforcement learning is a class of algorithms in which a goal-seeking *agent* seeks to complete a task by means of interaction with an *environment*. An agent is a controller that maps observed variables to actions. The learning process originates with the agent directly exploring an environment by means of trial-and-error. The environment communicates relevant information about its dynamics to the agent by means of a *state* signal, which the agent then employs to perform some *action*. The environment updates its current state based on the action and computes a numerical reward that communicates the immediate benefit of the given action. This process repeats iteratively such that, over time, the agent improves its *policy* (the means by which decisions

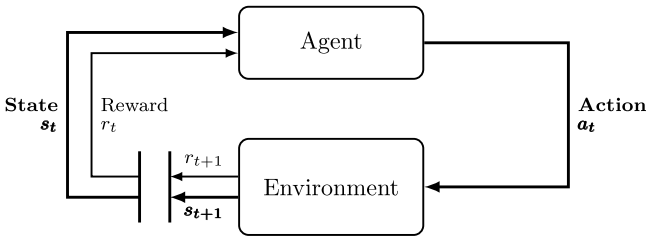



**Fig. 3.** The agent–environment process in a Markov decision process.
*Source:* Reproduced from Sutton and Barto, Ref [48], p.48. Figure 3.1.

are accomplished) by seeking to maximize the reward signal. In many cases, terminal conditions exist that cause the learning process to cease. In these cases, the environment typically resets to an initial configuration, and the process begins anew. These are identified as *episodic* tasks, where each episode signifies an attempt by the agent to solve a problem. A schematic for the high-level process of an RL agent is depicted in Fig. 3. This diagram highlights the three signals that enable communication between the agent and environment at time steps $t$ and $t+1$: state, action, and reward. While the RL literature prefers the terms *agent*, *environment*, and *action*, these expressions are analogous to the more common engineering terms *controller*, *controlled system* (or *plant*), and *control signal* [48].

Without external supervision, the agent uncovers complex dynamical structures that inform decision-making. This procedure is characterized as a Markov Decision Process (MDP), which involves both feedback and associative actions, and are a classical formalization of sequential decision-making problems [48]. The MDP is the idealized mathematical form of the problem, and allows for theoretical analysis of the RL process. In the infinite-horizon discounted case, the MDP is formulated by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, q_0, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are the sets of all possible states and actions, respectively, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state-transition probability distribution, $r : \mathcal{S} \to \mathbb{R}$ is the reward function, $q_0$ is the distribution of the initial states $\boldsymbol{s_0}$, and $\gamma \in [0, 1]$ is the discount factor [49].

For a problem to be accurately cast as an MDP, each state must satisfy the *Markov property*, which requires that future states depend only upon the current state, and not on the series of events that preceded it [48]. In many practical applications, only partial information is available, and the agent receives a subset of all environmental data. This signal is denoted the "observation" and serves as an analog to the state signal; such 'reduced' information procedures are labeled Partially Observable Markov Design Processes (POMDPs). The delineation between state and observation is important in POMDPs because it reinforces the notion that the agent is acting on incomplete information. However, this investigation assumes a fully observable MDP and, thus, for simplification, the observation $\boldsymbol{o_t}$ is represented as the state $\boldsymbol{s_t}$, and no such distinction is necessary.

An agent seeks to develop a policy that maximizes future reward by balancing immediate and future returns. This balance is formalized by defining the expected return, $G_t$, as the sum of future discounted rewards, i.e.,

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \tag{8}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the extent to which the agent prioritizes immediate versus future rewards. If $\gamma = 1$, all future rewards are considered equally; if $\gamma = 0$, only the immediate reward is included. The definition of expected return leads to the formalization of the *value* function. The value function is estimated in nearly all RL algorithms and informs the agent of the quality of a particular state. For an MDP, the *state-value function*, $V_\pi(\boldsymbol{s_t})$, is defined

as,

$$V_\pi(\boldsymbol{s_t}) = \mathbb{E}_\pi\left[G_t\right] = \mathbb{E}_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r(\boldsymbol{s_k})\right] \tag{9}$$

where, assuming the agent follows policy $\pi$, $\mathbb{E}_\pi\left[G_t\right]$ is the expected value of a random variable at any time step $t$. It is similarly useful to estimate the value of a state–action pair. The *state–action value function*, denoted $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$, computes the value $V_\pi(\boldsymbol{s_t})$ of taking action $\boldsymbol{a_t}$ at $\boldsymbol{s_t}$, and subsequently following the policy $\pi$. This function, $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$, is closely related to the state-value function, Eq. (9), and is computed as,

$$Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t}) = \mathbb{E}_\pi\left[G_t\right] = \mathbb{E}_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r(\boldsymbol{s_k}, \boldsymbol{a_k})\right] \tag{10}$$

where state-value and state–action-value are related by noting the distribution to which the action $\boldsymbol{a_t}$ belongs,

$$V_\pi(\boldsymbol{s_t}) = \mathbb{E}_{\boldsymbol{a_t}\sim\pi}\left[Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})\right] \tag{11}$$

The functions $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ and $V_\pi(\boldsymbol{s_t})$ differ in the action given that $\boldsymbol{a_t}$ in $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ is not necessarily sampled from policy $\pi$. In practice, many modern algorithms estimate these value functions using neural networks. Alternatively, rather than employing $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ or $V_\pi(\boldsymbol{s_t})$ to represent value information directly, some RL methods, such as the one employed in this research effort, are concerned with estimating the value of a state–action pair compared to average. In these approaches, a relative value, termed the *advantage* function, is computed as the difference between the state–action-value function and the state-value function [50],

$$A_\pi(\boldsymbol{s_t}, \boldsymbol{a_t}) = Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t}) - V_\pi(\boldsymbol{s_t}) \tag{12}$$

The advantage function is positive when $\boldsymbol{a_t}$ produces a more desirable outcome than an action sampled from the current policy $\pi$, and is negative when $\boldsymbol{a_t}$ produces a less desirable outcome.

Policy optimization methods seek to directly 'learn' a parameterized policy, $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$, where $\boldsymbol{\theta}$ represent a policy parameter vector. In contrast to value optimization methods, such as $Q$-Learning and DQN, policy optimization methods are well suited to problems with a continuous action space. The ability to incorporate continuous state and action spaces offers significant advantage in complex control tasks that suffer from discretization and are more extendable to higher-dimensional dynamical models. A particularly fruitful branch of RL research emerges from a class of hybrid algorithms, identified as actor–critic methods. These approaches seek both the policy (i.e., *actor*) and the value (i.e., *critic*) functions, where both actor and critic are typically represented as neural networks. Common actor–critic schemes include Advantage Actor Critic (A2C) [51], Deep Deterministic Policy Gradient (DDPG) [52], Twin Delayed DDPG (TD3) [53], Soft Actor Critic (SAC) [54], Trust Region Policy Optimization (TRPO) [49], and Proximal Policy Optimization (PPO) [55]. This class of algorithms are of particular recent interest, and PPO is employed in this investigation.

### 3.1.4. Policy gradient methods

Policy gradient methods are a subclass of policy optimization algorithms that utilize gradient descent to optimize policy parameters. In these approaches, during training, an action is sampled from a diagonal Gaussian distribution (i.e., a multivariate normal distribution) $\boldsymbol{a_t} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})$, where the mean values are computed by the agent. Including variance in the selected actions during training allows the agent to more thoroughly explore the environment and action spaces. The mean values are typically produced by an actor neural network that employs a state or observation as an input and outputs the mean of each action variable. The actor NN employed in this investigation appears in Fig. 4. The standard deviation for the resulting distribution is typically computed in one of two ways. First, the standard deviations may be included as outputs from the neural network, so that the actor distribution is entirely state-dependent. Alternatively, a more common approach in recent implementations is to maintain a vector of log standard deviations such that the variance is not a function of the state. The latter technique is employed in this investigation. After training is complete, the standard deviations are no longer incorporated, and the actor neural network is employed directly as a deterministic controller.

The agent's policy is learned by optimizing the weights of the actor NN, denoted $\boldsymbol{\theta}$, to maximize performance. As detailed by Sutton and Barto [48], performance is quantified by using the maximization of the value function as an objective function,

$$J(\boldsymbol{\theta}) = V_{\pi\boldsymbol{\theta}}(\boldsymbol{s_0}) \tag{13}$$

where $\boldsymbol{s_0}$ is a non-random initial state. With the objective function in Eq. (13), it follows that the update gradient for the actor is given by,

$$g = \nabla_{\boldsymbol{\theta}}\left[V_{\pi\boldsymbol{\theta}}(\boldsymbol{s_0})\right] \tag{14}$$

The gradient in Eq. (14) is not typically computed directly. In policy gradient methods, direct computation results in prohibitively noisy gradients that prevent learning. Alternatively, using the critic network in actor–critic methods results in less noisy estimates, but introduces a significant bias into the computation. Therefore, most algorithms estimate the gradient using a methodology detailed by Schulman et al. [50]. A common policy gradient estimator is,

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})\hat{A}_t\right] \tag{15}$$

with the corresponding policy gradient loss function,

$$L^{PG}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t\left[\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})\hat{A}_t\right] \tag{16}$$

Many of the earlier policy gradient algorithms directly optimized Eq. (16) (e.g., REINFORCE [56]). However, the variance in gradient estimation frequently leads to destructively large policy updates in these strategies. For example, with a stochastic policy, an unexpected experience can yield extremely steep gradients, resulting in updates that render certain actions more or less probable by orders of magnitude. Applying such updates destabilizes the learning process and prohibits learning an optimal policy [55].

Repeatedly optimizing Eq. (16) leads to potentially destructive policy updates, so Schulman et al. introduce a "surrogate" loss function that avoids large policy updates and produces training that follows monotonic improvement. The resulting algorithm is denoted Trust Region Policy Optimization (TRPO) and forms the basis for many productive modern RL algorithms [49]. Prior to TRPO, gradient-based methodologies never outperformed dynamic programming on continuous control tasks. However, by including limiting bounds on policy updates, TRPO produces powerful and complex policies that outperform dynamic programming for many problems [49]. In TRPO, the agent samples actions from a policy distribution $\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$. With the state–action–reward trajectory generated under $\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$, a new policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})$ is optimized that, in turn, becomes the updated $\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$. By creating two distributions, the change in policy is quantified by employing the Kullback–Leibler (KL) divergence, which measures the similarity between two probability distributions. A constraint is then introduced into the optimization scheme to insert an upper limit on the KL divergence between $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}_{\text{old}}}$, resulting in monotonic policy improvement. Together, the optimization scheme under TRPO becomes,

$$\begin{aligned} &\underset{\boldsymbol{\theta}}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[R_t(\boldsymbol{\theta})\hat{A}_t\right] \\ &\text{subject to} \quad \hat{\mathbb{E}}_t\left[\text{KL}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot|\boldsymbol{s_t}), \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s_t})\right]\right] \le \delta \end{aligned} \tag{17}$$

where $\delta$ is a limit on the change in policy distributions and the ratio,

$$R_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})} \tag{18}$$

is the probability of an action, given a certain state under a new optimized policy, divided by the probability of that action under the

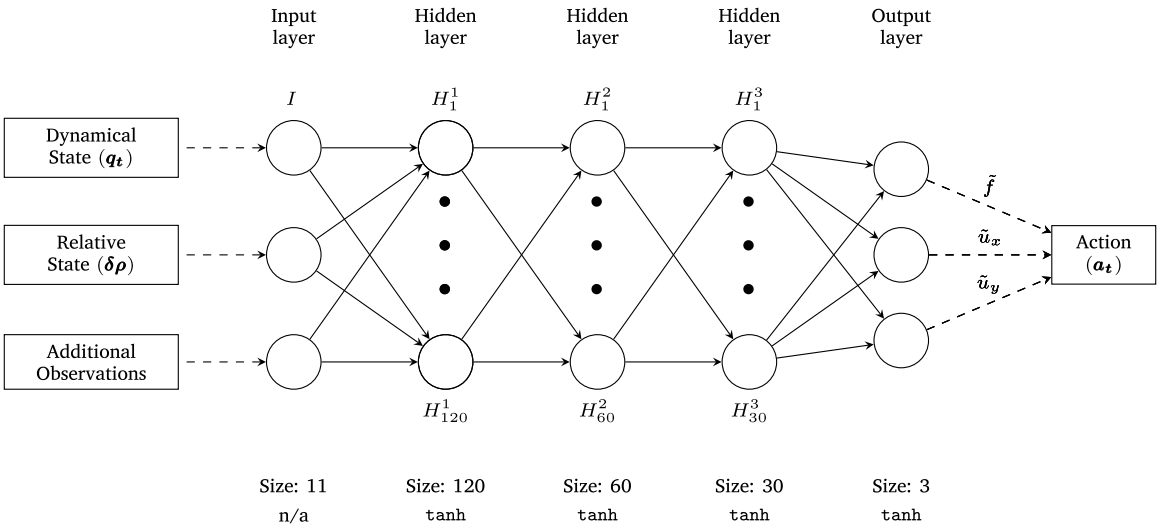



**Fig. 4.** Actor neural network employed in the proximal policy optimization algorithm.

previous policy distribution. The constraint inequality prevents prohibitively large updates by ensuring that the change in policy distribution remains within a specified bound. The ratio in Eq. (18) is most easily understood by considering the relative magnitudes of the numerator and denominator. If the optimization step causes an action $\boldsymbol{a_t}$ to become more probable, then $\pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t}) > \pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$, and so $R_t(\boldsymbol{\theta}) > 1$. Conversely, if the action becomes less probable, then $R_t(\boldsymbol{\theta}) < 1$.

While TRPO is successful in many challenging domains, there are several noted limitations. First, the algorithm itself is very complex and is, therefore, prone to implementation errors. Furthermore, TRPO involves a hard constraint in the optimization process due to the difficulty in selecting a single coefficient that transforms the KL-divergence constraint in Eq. (17) into a penalty [55]. Finally, TRPO is not compatible with tasks that include noise or parameter sharing.

### 3.1.5. Proximal policy optimization

Proximal Policy Optimization (PPO) is a simpler and more flexible alternative to achieve the same high performance as TRPO [55]. In formulating PPO, the KL divergence constraint in TRPO, Eq. (17), is replaced with a clipping factor that controls the maximum allowable update. The change in probability distribution is quantified by the ratio in Eq. (18). By introducing clipping, PPO optimizes a surrogate objective, that is,

$$L^{CLIP}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min\left(R_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(R_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{19}$$

where $R_t(\boldsymbol{\theta})$ is the probability ratio defined in Eq. (18). Multiplying $R_t(\boldsymbol{\theta})$ by the estimate of the advantage function forms the core of PPO: $R_t(\boldsymbol{\theta})\hat{A}_t$. Recall that the ratio $R_t(\boldsymbol{\theta}) > 1$ implies an action is now more probable under a new, optimized probability distribution. Next, the ratio $R_t(\boldsymbol{\theta})$ is multiplied by the advantage function $\hat{A}_t$, defined in Eq. (12), which measures the quality of a particular action as compared with the expected value. If the advantage is positive, an action is better than expected, and the optimization scheme enables this action to be more probable. Conversely, a worse-than-average action becomes less probable. Finally, the minimum value between the unclipped $R_t(\boldsymbol{\theta})\hat{A}_t$ and the clipped objective function between $1 \pm \epsilon$ forms a pessimistic lower bound for the unclipped objective. With $\epsilon = 0.2$ as a suggested value, an action is, at most, 20% more or less probable under PPO's optimization scheme. Analogous to the constraint in TRPO, clipping is included in PPO to eliminate the destructively large policy updates that are common in vanilla policy gradient schemes.

An alternative approach to PPO, rather than clipping the surrogate objective function, is to introduce an adaptive KL penalty coefficient [55]. The adaptive penalty approach is a hybrid between PPO and TRPO. Rather than clipping the objective function, the KL divergence is introduced as a penalty,

$$L^{KLPEN}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[R_t(\boldsymbol{\theta})\hat{A}_t - \beta \text{KL}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot|\boldsymbol{s_t}), \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s_t})\right]\right] \tag{20}$$

where a target KL divergence value, $d_{\text{targ}}$, is maintained by controlling a penalty coefficient, $\beta$. The penalty coefficient is updated by computing the expected value of the KL divergence,

$$d = \hat{\mathbb{E}}_t \left[\text{KL}\left[\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot|\boldsymbol{s_t}), \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s_t})\right]\right] \tag{21}$$

which is then compared to the target value to produce two possible updates,

$$\beta = \begin{cases} \max(\beta_{\min}, \frac{1}{1.5}) & \text{if } d < \frac{1}{2} d_{\text{targ}} \\ \min(\beta_{\max}, 1.5\beta) & \text{if } d > 2\, d_{\text{targ}} \end{cases} \tag{22}$$

while scalar parameters 1.5 and 2 are heuristically determined. However, Schulman et al. note that the algorithm is not sensitive to the specific choice of 1.5 and 2, or the initial value of $\beta$ [55]. The update rule for $\beta$ is simply understood as follows: when the probability distribution endures more change, the penalty coefficient is increased to encourage smaller policy updates. Conversely, when the distribution is less variable, the penalty is decreased to allow larger updates. Finally, $\beta_{\min} = \frac{1}{35}$ and $\beta_{\max} = 35$ heuristically bound the total possible change in the coefficient. While clipped PPO typically performs better than the KL penalty baseline [55], this research employs a KL penalty approach due to a dramatic increase in performance compared with vanilla-clipped PPO. Hees et al. also favor the KL penalty approach for 3D humanoid control tasks [24]. The specific implementation of PPO in this investigation is based on the open source work of Patrick Coady [57], and includes several other minor customizations to the core algorithm.

### 3.1.6. Implementation details

The PPO implementation employed in this investigation benefits from certain customizations that augment the core PPO functionality. These customizations build off previous contributions and enable effective neural network training. The architectures of actor and critic networks employed in this research are listed in Appendix, Table A.6.

The objective of the NN update process is to adjust the network parameters to maximize Eq. (18) for the actor (policy) network, and minimize Eq. (12) for the critic (value) network. In this optimization process, a form of gradient descent modifies the individual weights and biases of the neural networks based on the collected experience of the agent. Using a small quantity of data leads to poor gradient

estimation and can inhibit the learning process. To combat this phenomenon, a common approach in RL is to batch data from multiple episodes to ensure updates are based on accurate gradient information. Throughout training, each episode's actions, rewards, and observations are stored. Upon the completion of 20 episodes, this batch of data is collectively used to update the actor and critic neural networks. Twenty episodes between updates is selected to balance the quality of each update, which improves with more data, with the frequency of updates, though the precise batch size is problem dependent. As an *on-policy* RL algorithm, PPO can only update its current policy using episode data collected under that policy. As a result, after each update, the batched episode data is discarded and more episodes are conducted. To improve data efficiency, the update process is run multiple times (20 for the actor, 10 for the critic) before the batched data is discarded.

Episode data is converted into gradients using backpropagation [58]. This process applies the chain rule to determine the appropriate gradient for each network parameter in the neural network's hidden layers based on the error in the output layer. The selected gradient descent algorithm is then applied. At its simplest, gradient descent updates network parameters $\boldsymbol{\theta}^-$ via a step of length $\psi_{\boldsymbol{\theta}}$ along the gradient vector $\nabla_{\boldsymbol{\theta}} L^{KLPEN}(\boldsymbol{\theta})$,

$$\boldsymbol{\theta}^+ = \boldsymbol{\theta}^- + \psi_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} L^{KLPEN}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-} \tag{23}$$

The gradient descent algorithm Adaptive Moment Estimation (Adam) [59] is employed in this investigation. Adam, building off stochastic gradient descent, applies a bias-corrected moving average of the gradient to dampen any potentially noisy, individual updates.

This investigation's PPO implementation includes an adaptive learning rate for the actor network. Learning rates are initialized to 0.0001 for actor updates and 0.00025 for critic updates. To adjust the critic update speed based on the computed KL divergence penalty, Eq. (22), a scaling factor $\zeta$ is included,

$$\zeta = \begin{cases} \frac{2}{3}\zeta & \text{if } \beta > 30 \text{ and } \zeta > \zeta_{\min} \\ \frac{3}{2}\zeta & \text{if } \beta < \frac{1}{30} \text{ and } \zeta < \zeta_{\max} \end{cases} \tag{24}$$

where $\zeta_{\min} = 1/10$ and $\zeta_{\max} = 10$ bound the possible values of $\zeta$. When the actor is trained, the initial learning rate 0.0001 is multiplied by $\zeta$ to produce the learning rate for that training batch. An adaptive step size for Adam is suggested by Schulman et al. [55], and is similarly implemented in a six degree-of-freedom planetary landing problem [14]. The multiplier is helpful in situations where the KL divergence quickly changes and pushes $\beta$ outside its boundary conditions.

Input data scaling is a common practice for deep neural networks to ensure each input variable is treated equally. A common practice is to standardize all inputs to be mean zero and standard deviation one [44]. This standardization is accomplished by collecting data during training to maintain accurate mean and variance information for each input value. Prior to optimization of the actor and critic neural networks, input variable statistics are updated based on the current batch of experience. This process ensures each variable remains bounded and allows for observations with varying orders of magnitude.

### 3.2. Environment configuration

Properly formulating a reinforcement learning environment is critical to algorithmic performance. Researchers develop RL algorithms to be applicable to many types of problems, however, each environment must be specifically tailored to a particular application. It is especially important to design an environment such that the underlying assumptions are not violated in the RL process. In particular, the environment quantifies the problem of interest, the system dynamics, episode details, and the process to pass information back-and-forth between the agent and environment. In particular, as depicted in Fig. 3, the state, action, and reward signals quantify the communications process. Implementing an effective environment involves properly defining each signal. The environment must define the initialization and termination of episodes and must ensure that its computational footprint does not inhibit learning performance.

While both the selection and implementation of an appropriate RL algorithm is critical to the learning performance, so too is proper design of the RL environment. The environment represents the formulations for the state, action, and reward signals. The state vector, $\boldsymbol{s_t}$, communicates relevant information to the agent about the environment at a particular point in time. Hence, the state must be designed to accurately communicate information about the environment dynamics and subsequent flow. The action, $\boldsymbol{a_t}$, defines an agent's ability to alter that environment and must offer the agent sufficient control authority to 'learn' an effective policy. Lastly, the reward signal, $r$, is a scalar value that denotes the immediate positive or negative impact of a particular action. The selection of a reward function is arguably both the most difficult and most important function for design and is, thus, a critical element of this learning framework. Proper signal design is vital because even the most robust learning algorithm consistently falls short in an ill-designed environment. Hence, a proper quantification of positive and negative behavior, given the goals of the guidance framework, is crucial in achieving desirable outcomes.

#### 3.2.1. State signal

Under a Markov Decision Process (MDP), the environmental state at time $t$ ($\boldsymbol{s_t}$) must include all necessary past environmental information that impacts the future [48]. For the CR3BP, position, velocity, and spacecraft mass are together sufficient, since future states are predicted by numerically integrating the equations of motion specified in Eqs. (1)–(4). Hence, at every time step $t$, the dynamical state $\boldsymbol{q_t}$ is defined as,

$$\boldsymbol{q_t} = \begin{bmatrix} \boldsymbol{\rho}^{\text{agent}} & m \end{bmatrix} = \begin{bmatrix} x & y & \dot{x} & \dot{y} & m \end{bmatrix} \tag{25}$$

While $\boldsymbol{q_t}$ alone is sufficient to satisfy the Markov property in this planar problem, the agent performance is greatly enhanced by augmenting the dynamical state, $\boldsymbol{q_t}$, with additional variables to form the state signal, $\boldsymbol{s_t}$. In the PPO formulation, the actor and critic networks receive the complete state signal as inputs, as depicted in Fig. 4. Hence, both the policy and value functions are dependent on the selection of additional variables. Since this problem involves an agent learning to track a reference solution, relative position and velocity are essential to the agent's performance and its ability to generalize to nearby motion. The relative information is computed simply as,

$$\boldsymbol{\delta\rho} = \boldsymbol{\rho}^{\text{agent}} - \boldsymbol{\rho}^{\text{ref}} = \begin{bmatrix} \delta x & \delta y & \delta \dot{x} & \delta \dot{y} \end{bmatrix} \tag{26}$$

where $\boldsymbol{\rho}^{\text{agent}}$ is the position and velocity of the agent at some time step, and $\boldsymbol{\rho}^{\text{ref}}$ is the position and velocity of the nearest neighbor along the given reference trajectory path. Here, "nearest" is defined as the state along the reference with the lowest L2 norm for the relative position and velocity. Note that this definition of "nearest" does not include time and, hence, is a *normal* rather than an *isochronous* correspondence. If the reference path includes a set of $n$ discrete points, $\mathcal{R}^{\text{ref}}$, then the nearest state $\boldsymbol{\rho}^{\text{ref}}$ is defined as,

$$\boldsymbol{\rho}^{\text{ref}} \in \mathcal{R}^{\text{ref}} \quad \text{s.t.} \quad k = |\boldsymbol{\delta\rho}| = \sqrt{\delta x^2 + \delta y^2 + \delta \dot{x}^2 + \delta \dot{y}^2} \quad \text{is minimal} \tag{27}$$

The scalar value $k$ is a function of both the position and velocity deviation. This relative state information $\boldsymbol{\delta\rho}$, along with the dynamical state $\boldsymbol{q_t}$ and other optional additional observations, form the complete state signal,

$$\boldsymbol{s_t} = \begin{bmatrix} \boldsymbol{q_t} & \boldsymbol{\delta\rho} & \text{additional observations} \end{bmatrix} \tag{28}$$

of dimension $9 + j$, where $j$ is the number of optional additional observations that are incorporated. Recall that, for a fully observable MDP, the state $\boldsymbol{s_t}$ and observation $\boldsymbol{o_t}$ are interchangeable. The elements of the state signal must communicate sufficient information about the

environment to enable the actor and critic networks to accurately characterize the system dynamics.

The additional observations are problem-dependent and are, thus, included here as optional parameters. Since this investigation involves the CR3BP dynamical model, including some dynamical information in the reward signal is advantageous for learning performance. In particular, the Jacobi constant, defined in Eq. (7), communicates energy deviations to the actor and critic networks. At each time step, the Jacobi constant for $\boldsymbol{s_t}$, denoted $C_{s_t}$, is computed for a particular state, and then combined with the Jacobi constant value from the reference trajectory, $C_{\text{ref}}$, to form the additional observations in Eq. (28). The complete state vector in the CR3BP environment is then defined as,

$$\boldsymbol{s_t} = \begin{bmatrix} \boldsymbol{q_t} & \boldsymbol{\delta\rho} & C_{s_t} & C_{\text{ref}} \end{bmatrix} \tag{29}$$

The Jacobi constant is included to communicate the importance of energy in the CR3BP, however, omitting these optional observations does not prohibit convergence.

### 3.2.2. Action signal

In any MDP, an agent influences the environment by means of actions. For a low-thrust spacecraft, the action takes the form of a thrust magnitude and direction. While this action does not instantaneously alter the environmental state, its impact is realized in the numerical propagation that occurs between time steps. In this case, the action is identified by the actor network, which outputs both a thrust magnitude, $\tilde{f}$, and the vector components representing thrust direction, $(\tilde{u}_x,\ \tilde{u}_y)$, as depicted in the output layer of the actor network in Fig. 4. During the training phase, the network outputs the mean value of each action parameter and uses these in conjunction with a derived variance to create a normal distribution for each value. The mean is essentially the agent's best guess for the action given a particular observation, and the variance is included to encourage exploration. Miller, Englander, and Linares employ a similar action definition for interplanetary trajectory generation [60]. As in all policy optimization RL methods, over the course of training, the output of the network approaches an optimal policy. Once fully trained, exploration is no longer necessary, so the mean values are used directly to form a deterministic controller.

For a neural network controller, the raw value of the resulting action is governed by the selected activation function in the output layer of the network. The activation function employed in this investigation is tanh and, therefore, action values are bounded by $[-1, 1]$ (Fig. 2) and are scaled to reflect actual low-thrust values. Let 'tilde' denote raw value output by the network such that $\tilde{f},\ \tilde{u}_x,\ \tilde{u}_y \in [-1, 1]$. First, the thrust magnitude is re-scaled by the maximum total allowable nondimensional thrust,

$$f = \frac{\tilde{f} + 1}{2} f_{\max} \in [0, f_{\max}] \tag{30}$$

and the thrust directions are combined and normalized to form a unit vector. With this, the action is delivered as,

$$\boldsymbol{a_t} = \begin{bmatrix} f & u_x & u_y \end{bmatrix} \quad \text{such that} \quad \hat{u} = [u_x\ \ u_y] = \frac{[\tilde{u}_x\ \ \tilde{u}_y]}{\sqrt{\tilde{u}_x^2 + \tilde{u}_y^2}} \tag{31}$$

While parameterizing thrust as a unit vector/magnitude is straightforward, a potential drawback is an equation of constraint that is unknown to the controller, i.e., thrust direction is normalized after neural network evaluation. While it seems appealing to reformulate the low-thrust parameterization to eliminate this constraint, note that including an angle as an output value for any bounded activation function results in a critical discontinuity in the available action space and, therefore, in the gradient of the action with respect to the observations. This discontinuity occurs because, once re-scaled to a range $[0, 2\pi]$, the agent cannot perform an update step to push the output angle past either end bound. Hence, while parameterizations that include angles are potentially beneficial for other applications, such as trajectory design [38] and targeting [61], the bounded action implies that an alternate approach is required for this application.

An alternative low-thrust parameterization that has been applied to PPO is empowering the agent to command the thrust in each direction independently, such that each direction is allowed to employ the maximum allowable thrust [33]. While this approach avoids the discontinuity issue associated with angles, the drawback is that a physical engine has an associated total maximum thrust, but does not generally possess a practical limitation on the thrust direction of individual vector components. For the action to be more reflective of a physical engine, the magnitude/unit vector formulation, detailed here, separates thrust magnitude and direction in the action output. While the external equation of constraint that accompanies this strategy causes repetition in possible actions, it does not prohibit convergence to an effective policy.

### 3.2.3. Reward signal

Mission scenarios included in this investigation involve multi-body orbit transfers. These problems task a spacecraft to follow a ballistic 'reference' trajectory that asymptotically terminates at a periodic 'arrival' orbit. The environmental reward is designed to measure 'nearness' to the reference trajectory or arrival orbit as a scalar value. This nearness function is modeled as an exponential so that the reward grows rapidly as the agent's state nears the reference in both position and velocity. In this formulation, after the nearest neighbor along the reference is determined, the agent is rewarded for a thrusting plan such that the distance to the nearest state at the next time step is minimized. The reward function is then multiplied by a scaling term, $\eta$, to increase the reward over time for the reference solution. Reward is computed at each time step when the relative position and velocity are both less than an upper bound, denoted $\delta r_{\max}$ and $\delta v_{\max}$, respectively. If the deviation exceeds a maximum threshold, a penalty is applied, and the episode is terminated. Together, the reward function is defined as a piecewise function,

$$r = \begin{cases} \eta e^{-\lambda k} & \sqrt{\delta x^2 + \delta y^2} < \delta r_{\max} \ \text{ and } \ \sqrt{\delta \dot{x}^2 + \delta \dot{y}^2} < \delta v_{\max} \\ p & \text{deviate from reference or impact} \\ b & \text{arrival condition met} \end{cases} \tag{32}$$

where $\lambda$ is a scaling factor that increases the gradient of the reward, $k$ is defined in Eq. (27) as the relative distance (in position and velocity) to the nearest point along the reference, and $p$ is a penalty for deviating from the reference or impacting the primary or secondary body. Finally, the scaling term $\eta$ is evaluated as,

$$\eta = \frac{i}{n}\xi + 1 \tag{33}$$

where $i$ is the index of the reference trajectory state, $n$ is the size of the set of states along the reference trajectory, $\mathcal{R}^{\text{ref}}$, and $\xi$ is a tuning variable to adjust the rate at which the reward increases along the reference path. If the nearest neighbor is along the arrival orbit and not the reference trajectory, then $\eta$ is assumed to be a maximum value $\eta_{\text{arrival}} = \eta_{\max} = \xi + 1$. Formulating the reward signal to be at a maximum value when the agent reaches its target encourages the agent to fully complete the given transfer. For error thresholds of $\delta r_{\max} = 8000$ km and $\delta v_{\max} = 35$ m/s, $\eta$ yields a maximum value $\eta \approx 0.04$, which implies that $\eta \in [0, 0.04]$. A value of $\eta > 0.04$ indicates a penalty is always applied, however, the penalty may still be applied in the case that position or velocity individually violates a threshold.

The reward function employed in this investigation differs from that in Miller and Linares [33] by removing time from the equation. In their formulation, a spacecraft is rewarded for arriving in a periodic orbit at a specific time. While requiring a matching time is important for many applications, such as rendezvous, other scenarios do not warrant this constraint. With a time-autonomous reward function, the agent returns to a reference trajectory, without penalty for a slightly longer or shorter transfer time.

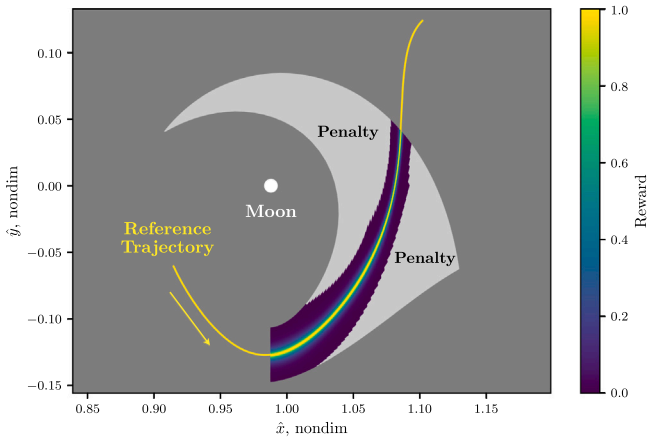



**Fig. 5.** Motion nearby a reference trajectory originating at the plane defined by $x = 1 - \mu$. Perturbations are introduced in $y_0$, and then propagated without thrust. Each state is colored based on the reward function defined by Eq. (32), where $\lambda = 100$, $\eta = 1$, and the maximum deviations in position and velocity are 8000 km and 35 m/s, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The measurement of nearness in the reward function, as detailed in Eq. (32), is visualized in Fig. 5. To illustrate the region of high reward surrounding the reference, perturbations are introduced in $y_0$ at the point where the trajectory crosses the $x = 1 - \mu$ plane. As each of these perturbed states is propagated forward in time, their deviation off the reference is visualized by the reward colormap. Once deviation beyond the threshold occurs, the trajectory is colored light gray to denote areas where a penalty is imposed. Due to the exponential term in Eq. (32), high reward exists solely in the region immediately surrounding the reference. Hence, to continue accruing reward, the agent is encouraged to maintain close proximity to the reference path.

A notable limitation of this time-autonomous approach is a reference trajectory that includes a departure periodic orbit. In this case, the reward function, as defined here, causes an agent to learn to maximize future reward by stationkeeping about the departure orbit rather than proceeding along the reference. To combat this behavior, the variable $\eta$ is leveraged in Eq. (32) to encourage the agent to continue along the reference, and discourage the initial periodic behavior. Like other optimization methods, abundant local minima encourage an agent to converge to sub-optimal behavior. Including $\eta$ discourages this behavior, but does not entirely eliminate the possibility of sub-optimal convergence and return to the stationkeeping local minimum.

#### 3.2.4. Nearest neighbor searching

The computation of the relative state for the reward and state signals presents some practical challenges in implementation. First, the nearest reference state must be selected from a discrete set of states along the reference path, $\mathcal{R}^{\text{ref}}$. For an accurate assessment of nearness, the trajectory must include a large number of states. However, since the reward is computed at every time step, a brute force search through $\mathcal{R}^{\text{ref}}$ is computationally infeasible. To ease this computational burden, the nearest neighbor search is instead conducted by traversing through a K-dimensional tree (KD-Tree). A KD-Tree is a data structure frequently used for data clustering in unsupervised learning applications. This specialized type of binary search tree reduces the algorithmic complexity of the nearest neighbor problem from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$, a significant improvement when $n$ is large. In this investigation, Scikit-learn's `neighbors.KDTree` implementation is employed to facilitate the nearest neighbor search process [62]. A similar approach is successful for autonomously locating neighbors in higher dimensional Poincaré maps [63,64]. This application differs in that the neighbor search is conducted for only a single state, rather than intersections from two discrete sets.

In addition to time complexity improvements, approaching the nearness function from an unsupervised learning perspective allows for the inclusion of additional dimensions at no cost to algorithmic complexity. This functional extendability allows for a simpler transition of this guidance framework to higher-dimensional dynamical models. However, when including multiple variables in the nearness metric, the KD-Tree approach dictates that all variables are condensed into a single norm function. Thus, the process is more complex if the variables are scaled differently, since the norm is then biased toward the results with larger absolute values. This drawback is addressed by scaling all variables to, approximately, the same order of magnitude. This investigation demonstrates that nondimensional position and velocity in the CR3BP are close in magnitude and do not demand re-scaling, however, if units are dimensional, or if additional variables such as time or an angle are included, the individual variable scaling issue requires re-assessment.

### 3.3. Episode details

The RL agent is trained over 150,000 finite-horizon episodes. The user-specified hyperparameters in the environment and the RL algorithm are summarized in Appendix, Table A.5. During each episode, the agent attempts to return to the reference trajectory and complete the given transfer scenario. Episodes are terminated when the agent diverges from the reference, or successfully reaches a specified arrival criteria. Over the course of training, the agent gradually improves at performing the task, until it consistently reaches the arrival condition. The overall trend in training over time is analyzed to understand performance, however, the specifics for the training protocols are most easily understood at the episode level.

Episodes begin by selecting a random initial state along the departure periodic orbit from a uniform distribution, and introducing a perturbation in position and velocity; initial mass is held constant. During training, $3\sigma_r = 1000$ km and $3\sigma_v = 10$ m/s model initial deviations. The perturbation is sampled from two normal distributions (position and velocity), where each component of $\boldsymbol{\rho}_0$ is perturbed individually. The averages across the perturbation distributions are all zero, and $\sigma$ is selected based on the desired disturbance for a particular simulation. As the L2 norm of the sum of two independent, normally distributed random variables, the scalar perturbation magnitude $k$ follows a chi distribution rather than a normal distribution [65]. This lower-fidelity error model is intended to encompass a wide range of deviation causes. Orbit Determination (OD) navigation errors on the order of $3\sigma = 1$ km and 1 cm/s [66] provide a useful comparison metric for the perturbation distribution. During training, initial deviations are three orders of magnitude larger than expected OD disturbances. Orbit determination errors are included as a comparison metric for the initial perturbation distributions, and are not implemented during simulations.

Once the initial state is generated, the departure orbit is no longer used in the simulation. Hence, to accrue reward over an episode, the agent must follow a particular reference trajectory as defined in the environment. After some disturbance is introduced into the environment, the agent computes an action, i.e., a thrust direction and magnitude. The equations of motion are then propagated within the environment for a particular time horizon ($\Delta t$). The specified $\Delta t$ between the actions is an important selection for the agent performance. If $\Delta t$ is too large, then the nonlinearities become more pronounced, and the agent is offered fewer opportunities for sufficient actions over an episode. However, if $\Delta t$ is too small, there is not sufficient time for the thrust direction to demonstrate a discernible impact on the system. For the examples included in this investigation, $t = 0.2$ nondim $\approx 20.87$ h strikes a balance between the extrema of being too large or too small. After propagating for $\Delta t$, the agent again selects a new action. Actions are introduced sequentially, after each time interval, until the agent deviates from the reference, impacts a planetary body, arrives at the target orbit, or reaches a maximum number of time steps.

**Table 2**
Characteristic quantities and low-thrust engine values in the Earth–Moon system.

| Earth–Moon characteristic quantities | | Low-thrust engine properties | |
|---|---|---|---|
| $\mu$, nondim | 0.012004715741012 | $f_{max}$, nondim | 0.04 |
| $l^*$, km | 384 747.962856037 | $I_{sp}$, s | 3000 |
| $t^*$, s | 375 727.551633535 | | |

The arrival condition is triggered when the nearest neighbor to the spacecraft is along the destination orbit and the relative position and velocity magnitudes are less than 100 km and 2 m/s, respectively. This definition of arrival is not intended to indicate that the spacecraft has reached a particular state. Rather, the given tolerances simply detect that the spacecraft has reached the vicinity of the arrival orbit. If a tighter tolerance for arrival is employed, many false negatives exist that eventually reach the given tolerance, but not within the specified maximum number of time steps. For example, relative position and velocity errors over time for a sample episode are depicted in Fig. 6, where annotations signify the time the arrival condition is satisfied given the noted tolerances. In the sample case, the arrival condition is triggered at 25.2 days when the relative position and velocity magnitudes along the arrival orbit drop below 100 km and 2 m/s, respectively. However, if 3 km and 0.07 m/s tolerances are instead selected, as depicted in Fig. 6, with a maximum episode time of 100 days, the arrival condition would not be met, and the episode would be falsely designated a failure. Past 25 days, the errors remain bounded, and reaching tighter tolerances is, in general, a result of noise rather than controller accuracy. Hence, 100 km and 2 m/s are heuristically selected as relative position and velocity magnitudes to ensure early arrival criteria detection.

## 4. Mission application: Libration point transfer

To illustrate the performance of the PPO-generated controller, several sample scenarios are considered. Due to the recent increased interest in cislunar space, the Earth–Moon CR3BP system serves as the basis for the sample cases. The specific characteristic quantities for this system along with the low-thrust CSI engine parameters are noted in Table 2. A comparison between this sample engine and various existing spacecraft is detailed in Table 1. The sample spacecraft possesses similar propulsive capability to the planned Psyche mission. Hence, the sample spacecraft is consistent with current low-thrust capabilities.

For the sample scenarios, planar transfers between orbits in the vicinity of the $L_1$ and $L_2$ libration points, with various geometries, serve as illustrative test problems for the proposed guidance framework. In particular, Lyapunov orbits at the same value of Jacobi constant are examined. With energy constrained, motion in the lunar vicinity is bounded in configuration space by a forbidden region, denoted the Zero Velocity Curves (ZVCs) [67]. Furthermore, the shared Jacobi constant value between the orbits indicates that heteroclinic transfers may be available. Heteroclinic transfers occur when manifold intersections create continuous $\Delta v$-free paths between two periodic orbits. These continuous trajectories are constructed by selecting an initial guess from manifold intersections on a Poincaré map and corrected using the methodology detailed by Haapala and Howell [11]. Heteroclinic transfers are one of the few cases in the CR3BP where globally optimal geometries may be identified and, thus, provide a useful test framework for the controller since no maneuvers are required.

While the agent is trained using only one reference trajectory, the controller's ability to generalize thrust histories to other geometries in the lunar region is also investigated. In reality, a variety of factors cause a planned path to shift in-flight. For example, onboard targeting yields trajectory corrections and a nearby solution is generated. In producing nearby transfers, it is often difficult to obtain any initial guess for the control history. As perturbations cause a spacecraft to deviate and maneuvers become necessary, a poor initial guess for control likely negatively impacts the performance of the targeter. To address this limitation, despite no training with other reference geometries, the ability for the proposed controller to generalize past experience is demonstrated. If a controller is only applicable to the particular reference it has seen, and the training process requires significant time and computational resources, then the practical uses of such a controller are limited in an onboard application. To test the controller performance for this generalization, other references and other transfers are examined. Finally, sample simulations are included to demonstrate a hybrid guidance approach where the neural network controller is applied in conjunction with a direct multiple shooting targeting scheme to impose additional constraints on the sample scenarios.

### 4.1. Scenario overview

Heteroclinic transfers between $L_1$ and $L_2$ Lyapunov orbits in the Earth–Moon CR3BP at an energy value $C = 3.124102$ are employed to demonstrate the training process, as well as to evaluate the performance of the guidance framework. The transfer scenario is illustrated in Fig. 7, where Fig. 7(a) depicts the periodic orbits, and Fig. 7(b) illustrates a continuous heteroclinic transfer from the $L_1$ Lyapunov orbit to the $L_2$ Lyapunov orbit. The agent is trained using this heteroclinic transfer, labeled reference 'A1', and is subsequently evaluated using other transfer geometries. All transfers at this energy level are labeled 'A'-transfers. Episodes are allowed a total of 100 time steps to reach the arrival criteria, or 86.97 days.

To accommodate variations in the controller training process, high-performance computing resources are leveraged to train many identically configured agents. Each agent is trained independently, and samples are not shared to ensure the on-policy assumption for PPO is satisfied. Training a large number of agents increases the chances of producing an effective controller that generalizes well, and demonstrates the prevalence of local basins of attraction. Once trained, the arrival criteria is employed to select an agent with desirable characteristics. Recall, *arrival* occurs when the spacecraft reaches the destination orbit, and relative position and velocity error magnitudes are less than 100 km and 2 m/s, respectively. Upon completing training, 2000 deterministic episodes are simulated to estimate the frequency with which each agent reaches the success criteria. Out of the 1275 trained agents, 40% are at least 90% successful and 25.7% are at least 99% successful. In practice, producing such a large number of agents is computationally challenging, and requires access to high performance computing resources. For this investigation, 1275 agents are produced to analyze training variance, however, only a small fraction of these are necessary to produce an agent that satisfies the mission criteria of this investigation.

### 4.2. Closed-loop controller performance

At any given point during training, a deterministic controller is available by simply removing variance from the agent's computed actions. Running an episode with the deterministic controller at various points in training, given a fixed initial condition, yields insight into the evolving improvement in the agent's policy. For this simulation, the initial conditions are generated by selecting an initial state along the $L_1$ Lyapunov departure orbit and introducing perturbations of 1106 km and 6.9 m/s. In reality, error is computed relative to the reference trajectory rather than the starting orbit, which adds a small amount of additional perturbation. For this sample case, the error relative to the reference is computed as 1108 km and 6.7 m/s. Without control, the resulting trajectory, as plotted in Fig. 8, immediately deviates from the reference and impacts the Moon in less than a week. This specific perturbation is not included at any point during the training phase. In many types of machine learning, it is important to separate training and validation data. In RL, this data separation is not as explicit as in supervised learning approaches, but it is nevertheless an important consideration when selecting a test case.

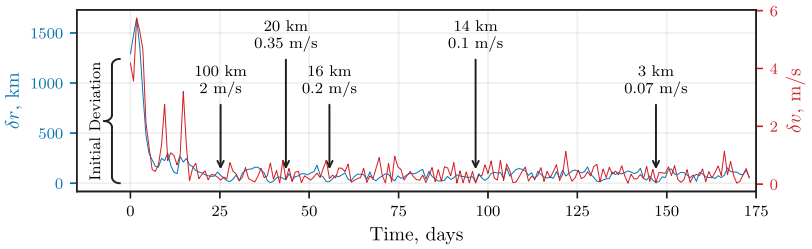
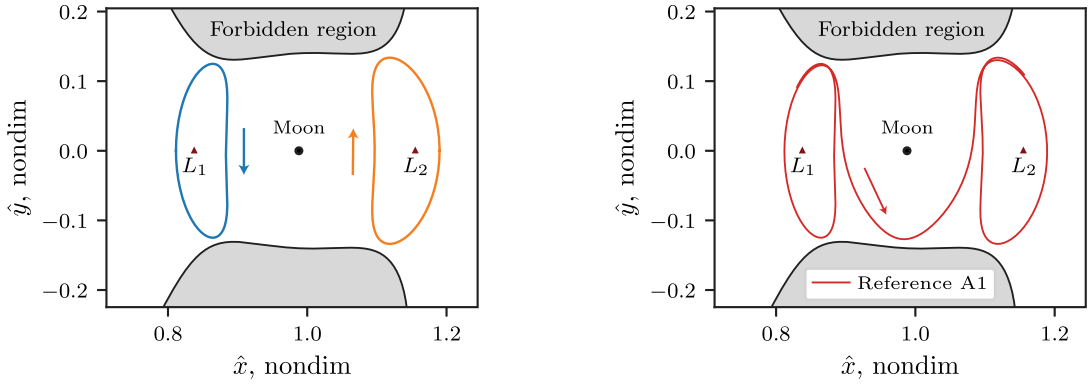



**Fig. 6.** Position and velocity deviations over time for a sample episode. Each arrow signifies the time when relative errors drop below the noted tolerances. Errors remain bounded once the destination orbit is reached, and smaller arrival tolerances may eventually be met given a longer time horizon.




(a) $L_1$ and $L_2$ Lyapunov orbits. (b) Heteroclinic transfer employed in training.

**Fig. 7.** Periodic orbits and heteroclinic transfer in the Earth–Moon system applied to the sample scenario. All motion at $C = 3.124102$, with the corresponding forbidden regions in gray.

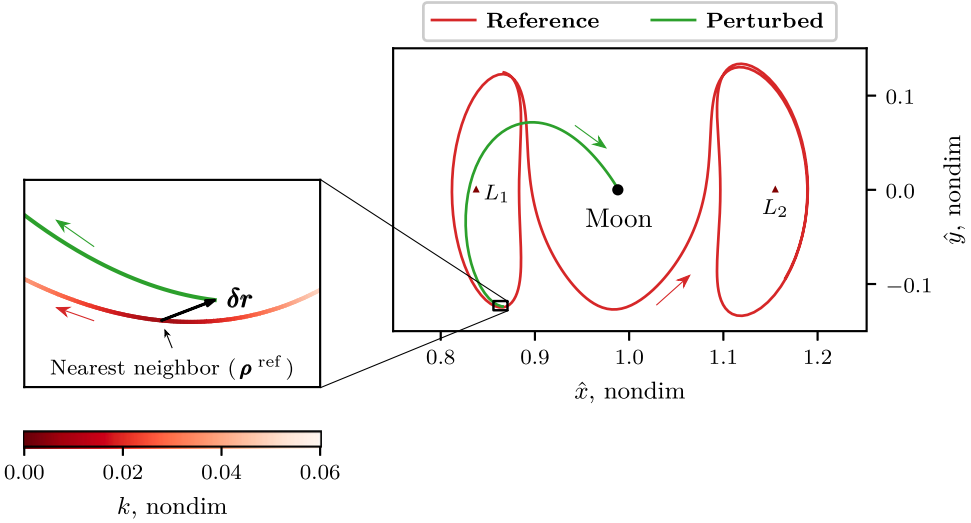
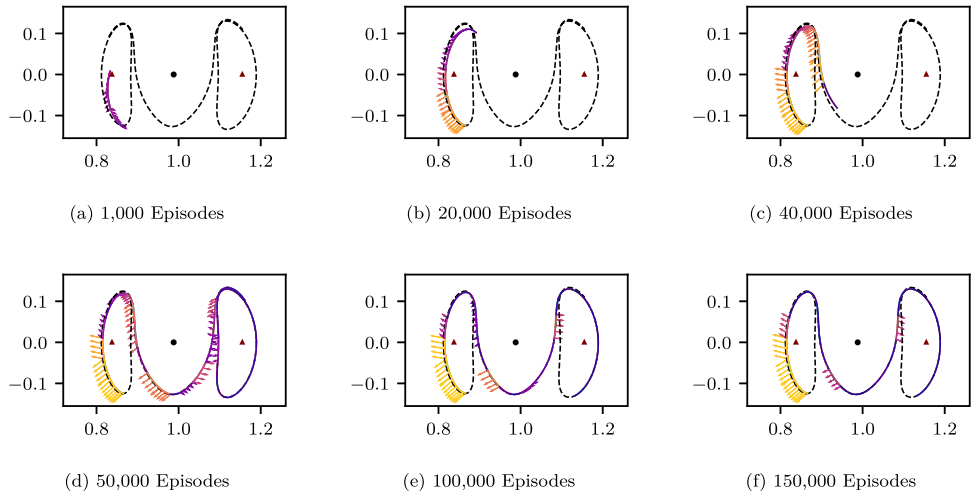



**Fig. 8.** Sample perturbed initial state that impacts the Moon in 6.4 days. Position perturbation is plotted in configuration space ($\delta r$), with magnitude 1106 km. States along the reference trajectory $\mathcal{R}^{\text{ref}}$ appear in the zoomed portion with the shade of red denoting the magnitude of the nearest neighbor distance to the perturbed state, $k$, defined in Eq. (27).

The control history produced by a deterministic controller at various stages in the training is plotted in Fig. 9. Arrows indicate the thrust magnitude and direction for a particular segment, where the length and color of the arrow corresponds to thrust magnitude. For visualization clarity, thrust values below a user-defined threshold cause the thrust direction magnitude indicator arrows to be omitted from visualization in Fig. 9. In this case, 25% is arbitrarily defined. When training begins, the agent's policy is determined by randomly initialized weights in the actor neural network, depicted in Fig. 4. As the number of episodes increases, the agent's ability to perform the given task gradually improves. After 1000 episodes, Fig. 9(a), the agent is not able to discern the correct thrust direction. By episode 40,000, the agent departs the Lyapunov orbit at approximately the correct location. After only 10,000 additional episodes, the agent completes the given transfer, though with substantially more thrusting than is necessary. Over the next 100,000 episodes, the agent gradually improves by reducing the amount of thrust applied. To illustrate the thrust reduction over learning, at episode 50,000, the agent performs the given task, but expends 0.38% of the available propellant. By episode 150,000, Fig. 9(e), expended propellant is reduced to 0.20%. Furthermore, this thrust reduction is visually apparent in the departure segment from the $L_1$ Lyapunov orbit.




(a) 1,000 Episodes (b) 20,000 Episodes (c) 40,000 Episodes

(d) 50,000 Episodes (e) 100,000 Episodes (f) 150,000 Episodes

**Fig. 9.** Deterministic agent over 150,000 episodes ($\hat{x}$-$\hat{y}$, nondim). Simulations (a – c) are penalized for diverging while (d – f) are terminated after successfully reaching the arrival criteria. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

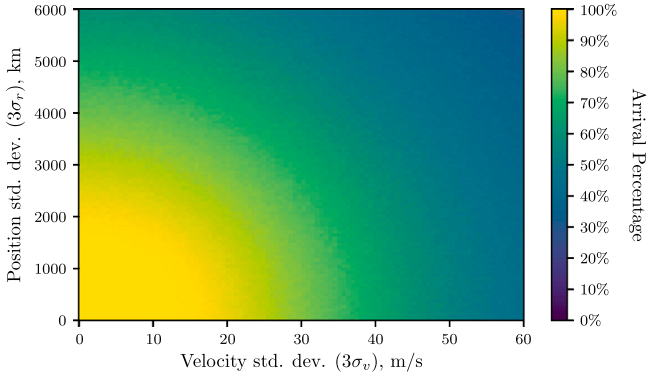



**Fig. 10.** Arrival percentage for training reference (A1) given various $3\sigma$ levels of initial error. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

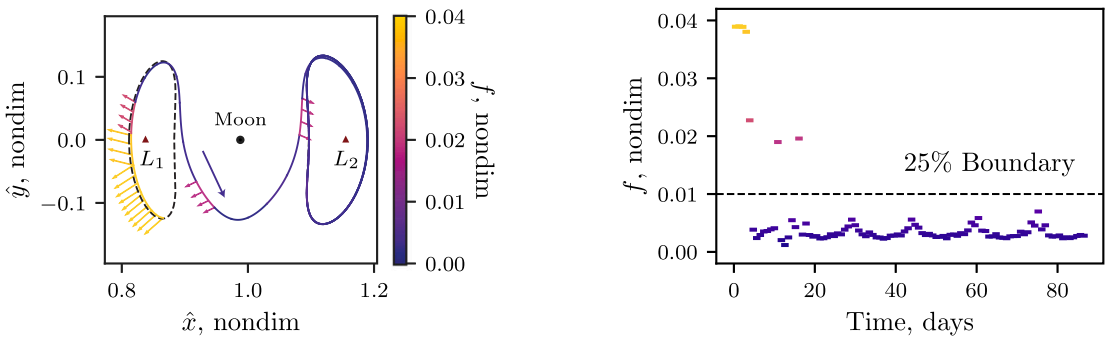
To analyze the performance of a particular controller, many initial conditions with various perturbations are generated and the agent is evaluated based on its resulting output control history. A simulation is considered successful if the agent avoids deviating from the reference and reaches the arrival criteria. The numerically computed arrival percentages for various levels of $3\sigma$ error in position and velocity are depicted in Fig. 10. For 10,000 combinations of position and velocity $3\sigma$ values, 5000 deterministic episodes are simulated using the sample agent and the training reference trajectory (A1). The results of the Monte Carlo analysis are colored based on the arrival percentage across the 5000 episodes. Based on expected levels of position and velocity error, Fig. 10 demonstrates the expectations for controller performance. For example, if all position and velocity errors are expected to be less than 1000 km and 10 m/s, respectively, then the bright yellow color at $[3\sigma_v = 10 \text{ m/s}, 3\sigma_r = 1000 \text{ km}]$ indicates that the controller is expected to reach the destination orbit in nearly 100% of cases (actual value is 99.5%). As expected, as the error increases, the controller becomes less successful. With large amounts of error, it is frequently unreasonable to return to a previous reference trajectory. Cases where the controller frequently fails to recover indicate error levels where a new reference trajectory is required.

The sample controller is more sensitive to perturbations in velocity than position, as depicted in Fig. 10. With perturbations sampled from $3\sigma_v = 0$ m/s and $3\sigma_r = 6{,}000$ km, the controller arrives successfully in 60.5% of the cases. Conversely, if $3\sigma_v = 60$ m/s and $3\sigma_r = 0$ km, the agent succeeds in only 46.7% of the cases. This sensitivity to velocity errors is consistent with other applications in cislunar space. For example, Davis et al. demonstrate the significant impact of velocity navigation errors in Near Rectilinear Halo Orbit (NRHO) maintenance [68].

### 4.3. Neural network-based guidance

For the perturbation in Fig. 8, the sample controller produces the control history in Fig. 11. With training completed, it is no longer necessary to terminate episodes upon reaching the arrival criteria. Without termination, upon completing the transfer, the agent performs orbit maintenance indefinitely about the $L_2$ Lyapunov orbit. The trajectory in Fig. 11(a) is identical to Fig. 9(f) since the resulting controller is simply the deterministic controller at the final episode, with the only difference being Fig. 9(f) terminates at arrival, and Fig. 11(a) does not.

Given the initial perturbation, as in Fig. 8, with the goal of returning to the original reference trajectory, a delay in response time renders the original geometry inaccessible. However, the trained neural network controller immediately outputs a control history that returns the spacecraft to its original path and successfully accesses the target $L_2$ Lyapunov orbit, as depicted in Fig. 11(a). Again, thrust magnitudes below 25% are omitted from visualization, but are still applied in the dynamical model. As expected, the majority of the spacecraft's thrusting occurs during the initial time intervals as the agent recovers from the introduced perturbation. Subsequent time intervals require much less propellant. Upon arrival in the destination $L_2$ Lyapunov orbit, the controller maintains the arrival geometry. The orbit maintenance is apparent in configuration space, i.e., in Fig. 11(a), as well as in the periodic sinusoidal thrust magnitude behavior, depicted in Fig. 11(b). Hence, the trained controller performs three functions: (1) recovers from a large initial deviation, (2) guides the spacecraft along a reference path, and (3) maintains stationkeeping about the destination orbit. Directly employing a neural network for guidance presents several practical challenges. While this simulation is deemed successful, it requires the spacecraft to be continuously thrusting, and offers no ability to include coasting segments. Furthermore, low-thrust engines typically possess a lower limit for thrust capability. The neural network is not aware of such limitations and, therefore, assumes the spacecraft engine implements any suggested maneuver. There are opportunities to include this sort of constraint in the training process, but this possibility is not investigated here. Depending on the lower-bound for the thrust level, it is possible for the controller to generate a new solution by simply not applying control values below the given threshold. Depending on the lower limit, the agent may still reach the arrival




(a) Trajectory and control history.

(b) Thrust magnitude history.

**Fig. 11.** Sample case where controller successfully overcomes an initial perturbation and follows a given reference trajectory. Thrust direction and magnitude are plotted in configuration space and colored based on thrust magnitude. For thrust values below an arbitrary threshold (25%), thrust direction indicators are omitted from the control history in (a). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

criteria, but the corresponding thrust history requires frequent large maneuvers and suggests a clearly suboptimal path. Hence, this thrust limitation is better addressed by either modifying the environment or augmenting the neural network guidance approach with an alternative guidance method. This investigation addresses thrust bounds by employing differential corrections to alter the neural network-generated solution.

### 4.4. Targeting hybrid guidance

Including mission criteria and constraints in the RL environment presents several challenges. While adding penalties to the reward function encourages an agent to avoid particular behavior, it may also cause sub-optimal convergence and unexpected behavior in policy optimization methods. For example, to encourage mass optimality, a thrust penalty may be included in the reward function. However, this additional term often produces agents that avoid the penalty by simply never thrusting. Furthermore, with constraints implemented as penalties, there is no guarantee that the final solution satisfies all objectives. An alternative approach is use of the neural network-generated solution as an initial guess for traditional guidance techniques such as targeting or optimization. This investigation demonstrates this hybrid process through a direct multiple shooting scheme.

Given the sample problem depicted in Fig. 8, a neural network controller recovers from the deviation with the control history in Fig. 11. While this solution satisfies the given problem, there are three notable limitations in the solution. First, the end condition is nearby the $L_2$ Lyapunov orbit, however, it may be desirable to ensure the trajectory terminates at the orbit with greater accuracy. Next, due to the relatively short time step between actions, the control history includes a large number of discrete thrusting segments (30). Finally, most of the segments possess small thrust magnitude values that pose practical implementation challenges for a low-thrust engine, and are likely unnecessary in solving the given problem.

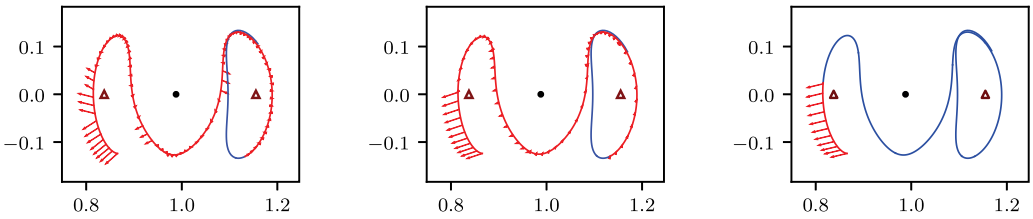
To address the limitations in the neural network-generated solution, a three-step multiple shooting targeting algorithm is employed. Using Fig. 11(a) as an initial guess, a final solution is produced through three targeting steps: (1) constrain the transfer end state to arrive on the stable manifold of the $L_2$ Lyapunov orbit, (2) reduce the number of discrete segments in the trajectory from 31 to 7, and (3) impose a minimum thrust threshold (50% of $f_{\max}$) to create coasting segments. The converged transfers following each targeting step are depicted in Fig. 12. The final solution, depicted in Fig. 12(c), possesses a single 3.6 day thrusting segment, followed by 20.26 days of coasting along the stable manifold of the destination orbit. Through the targeting process, the amount of expended propellant is reduced from 0.20% to 0.12% or, alternatively, the corresponding total equivalent $\Delta V$, computed by integrating the rocket equation, is reduced from 59.1 m/s to 34.3 m/s.

In a fully autonomous scenario, ensuring rapid targeting convergence is generally more important than fuel optimality. Analyzing the convergence basin corresponding to the final result, Fig. 12(c), lends insight into the accuracy of the neural network's startup solution. After reducing the total number of patch points, resulting in Fig. 12(b), a guess for the initial thrusting segment is produced. This guess possesses a single thrust direction and magnitude that is corrected to produce the final converged solution in Fig. 12(c). To analyze the accuracy and sensitivity of this guess, many combinations of thrust direction and magnitude are corrected via the targeter, with convergence characteristics plotted in Fig. 13. For this simulation, when convergence is achieved, all initial conditions result in the same final solution. In this sample scenario, the neural network-targeting hybrid solution (labeled "Step 2 Output" and visualized in Fig. 12(b)) produces an accurate initial guess, as depicted in Fig. 13. Furthermore, Step 3 of the differential corrections process converges in 12 iterations — substantially fewer than guesses near the edge of the basin.

Given one sample simulation, broad conclusions may not be drawn regarding the performance of this particular targeting scheme. In reality, many factors influence convergence characteristics, including the targeting approach, thrust parameterization choices, and tuning variables. Rather than analyze the specifics of this particular targeting methodology, the present investigation instead demonstrates where NN-based guidance may fit into existing targeting or optimization frameworks. In the sample case, targeting easily transforms the NN-generated trajectory into a feasible solution with a single maneuver.

### 4.5. Generalization to other references

If a spacecraft deviates significantly from its reference path, it is not always reasonable to return to the original trajectory. The process of generating a new transfer arc is accomplished from a variety of options, including leveraging dynamical systems theory, applying a numerical strategy, and/or employing differential corrections. The methodology for generating new reference transfers is not considered in this investigation. However, assuming a new trajectory has been constructed, an important test of the proposed controller is its ability to perform despite training with only one original path. For example, the hybrid targeting guidance approach produces the transfer depicted in Fig. 12(c), which no longer exactly matches the pre-computed reference in Fig. 7(b). If guidance is needed further along this new transfer, it must perform despite the geometry change. While neural networks, in general, offer a demonstrated ability to generalize, their performance is always tied to the training data set. Hence, if the new geometry is vastly different, a NN-based approach is frequently limited by its training experience.




(a) Step 1 (b) Step 2 (c) Step 3

**Fig. 12.** Targeter progression through three stages of targeting, where red denotes thrusting and blue denotes coasting ($\hat{x}$-$\hat{y}$, nondim). (a) The terminal state of the trajectory is constrained to be on the stable manifold of the $L_2$ Lyapunov orbit. (b) The number of discrete patch points in the solution is reduced from 31 to 7. (c) A 50% thrust threshold is applied to convert low magnitude thrusting segments to coast arcs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

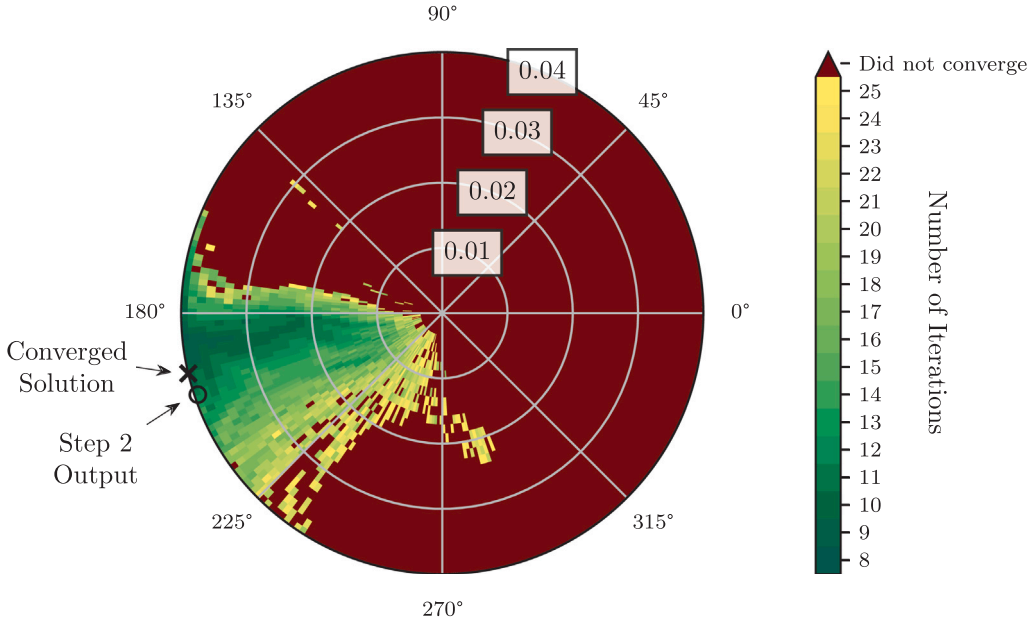
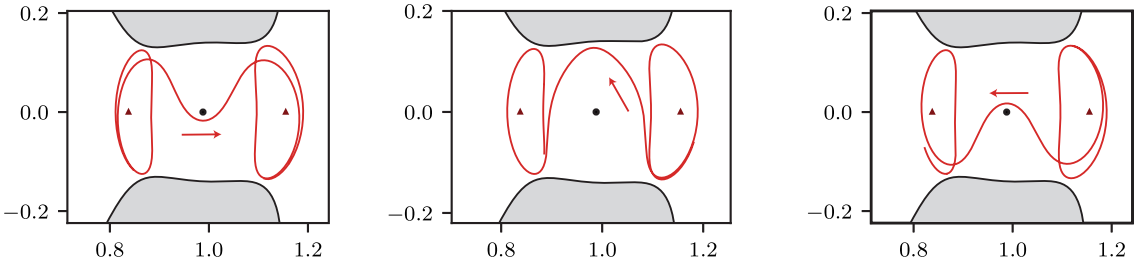



**Fig. 13.** Convergence across global spread of thrust initial conditions for targeting simulation in Fig. 12(c). In polar representation, the angle denotes the thrust vector angle measured from the rotating $\hat{x}$ axis, and radius denotes nondimensional thrust magnitude ($f$). Each angle/magnitude combination is colored based on the number of targeting iterations, with red denoting divergence or exceeding the maximum number of iterations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

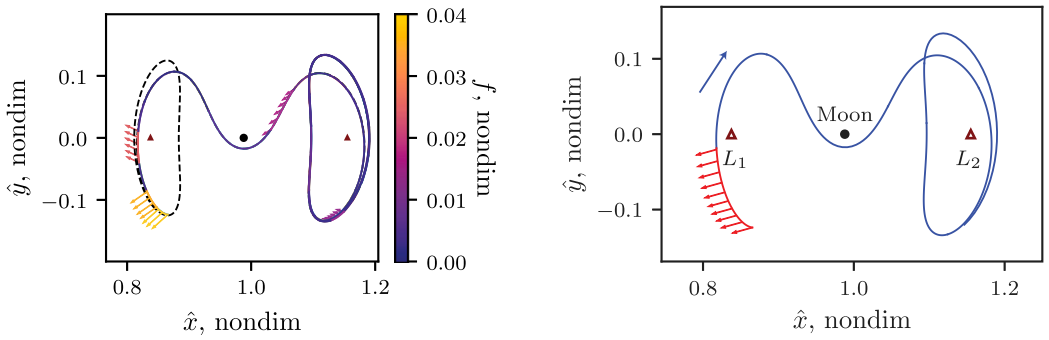



(a) Reference A2 ($L_1$ to $L_2$). (b) Reference A3 ($L_2$ to $L_1$). (c) Reference A4 ($L_2$ to $L_1$).

**Fig. 14.** Additional heteroclinic transfers between Lyapunov orbits at $C = 3.124102$ ($\hat{x}$-$\hat{y}$, nondim). Reference A2 (a) connects $L_1$ to $L_2$, and passes closer to the Moon than A1. References A3 (b) and A4 (c) connect $L_2$ to $L_1$, and serve as mirrored analogs to references A1 and A2.

To test the extendability of the PPO-generated controller, several new transfers are examined. First, a new path between the original $L_1$ and $L_2$ Lyapunov orbits is demonstrated via a second heteroclinic reference trajectory in Fig. 14(a). Next, the reverse of the previous example is also included, in which the agent originates in the $L_2$ Lyapunov orbit and attempts to track heteroclinic transfers to the $L_1$ Lyapunov orbit. These two new heteroclinic paths are plotted in Figs. 14(b) and 14(c).

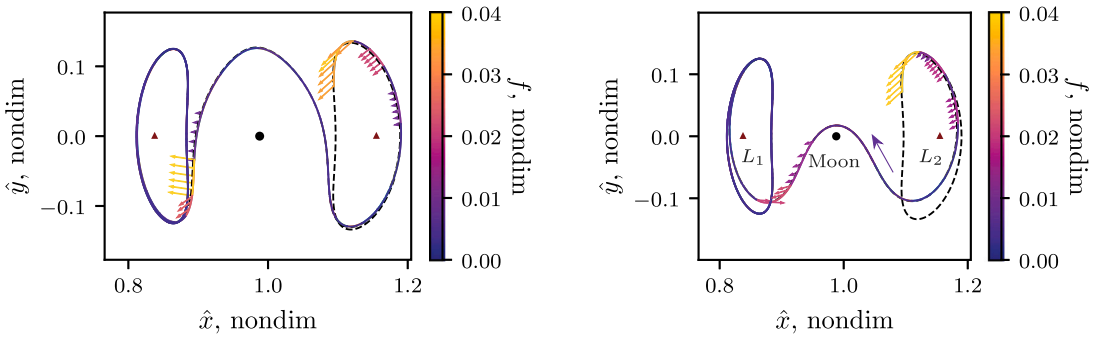
To evaluate the scenario where a new reference is generated in-flight, reference A2 in Fig. 14(a) is employed. Reference A2 is not included in the training process and, therefore, the controller must generalize its experience on the training reference to a different area of cislunar space. The new path is notably different from the original; the nearest distance to the Moon along references A1 and A2 are 34,546 km and 6725 km, respectively. Not only is this a large deviation in geometry,




(a) Neural network control history.

(b) Corrected single maneuver transfer.

**Fig. 15.** Resulting control history for previous sample deviation, plotted in Fig. 8, but using reference A2 from Fig. 14(a). Without additional training, the agent successfully tracks a new reference trajectory. Employing differential corrections, the neural network-generated solution (a) is transformed into a single maneuver solution (b).




(a) Control history for Reference A3

(b) Control history for Reference A4

**Fig. 16.** Sample computed control histories for the $L_2$ to $L_1$ reference trajectories plotted in Figs. 14(b) and 14(c).

significant nonlinearity is added due to the close proximity with the Moon. Hence, this example adds difficulty for the controller, not only in generalizing to new locations in space, but also demanding that the controller overcome more nonlinearity than was present in the training. Returning to the previous example, the perturbation in Fig. 8 is applied to the new reference geometry. The error from the perturbed state to its nearest neighbor along the new reference path is 860 km and 5.1 m/s. The agent generalizes its experience to the new reference trajectory, and the resulting control history and successful transfer are plotted in Fig. 15.

For realistic scenarios, since the agent's performance is dependent on training data, it is generally inadvisable to reuse an old agent for a drastically different scenario without training on the new geometry. However, examples where no additional training is conducted are included to illustrate the agent's ability to perform well in regions of space that were not originally explored. In particular, the environment is reversed from the previous example, that is, the agent is required to transfer from the $L_2$ Lyapunov orbit to the $L_1$ Lyapunov orbit along heteroclinic paths A3 and A4, plotted in Fig. 14. A single sample case is again examined to illustrate the neural network-generated solution. For an arbitrary perturbation, the agent is tasked to return to one of the new references, without the benefit of previous training. The controller completes these new transfers, with the resulting control histories plotted in Fig. 16. These examples demonstrate the RL controller's ability to perform well in challenging scenarios.

### 4.6. Monte Carlo results

A Monte Carlo analysis approach is applied to evaluate the sample controller's performance across different references. For this analysis, multiple levels of error are simulated for 50,000 trials each across the four sample reference trajectories. Initial position and velocity components are perturbed individually at varying orders of magnitude and larger than potential orbit determination navigation errors of $3\sigma = 1$ km and 1 cm/s, as detailed in Section 3.3. The proposed controller is tested with errors up to 2000 times the expected navigation error. The 1000× case corresponds to the error used in training, which represents the situation where a 1 km and 1 cm/s perturbation is introduced, and the error is propagated for an orbital period, or about 12.9 days. Testing the controller beyond the error levels introduced during training provides insight into performance limitations.

The results from the Monte Carlo analysis are summarized in Table 3, with failure rates plotted in Fig. 17. For the training reference A1, the controller is 100% successful for all errors less than the 500× scenario. However, as the error is increased, trials emerge where recovery does not occur. In the 1000× case, 0.5% of perturbations are not successfully recovered. For some of these examples, where the error bounds are $3\sigma = 1000$ km and 10 m/s, it is unreasonable to expect a return to the original motion, and these error levels may correspond to conditions where alternate options should be considered.

To illustrate the failure characteristics that occur for the 1000× case, deviations over time for 100 sample episodes are represented in Fig. 18. The red trajectories correspond to failures where the episode is terminated when either of the maximum deviation thresholds is violated, or if arrival conditions are not met within the maximum number of time steps. Episodes that reach the arrival criteria, in green, clearly maintain a small amount of error upon arrival to the target orbit. However, during the transfer phase, more variance is observed in the deviations. This variation is likely caused by the selection of a sub-optimal action which forces the agent to recover from additional error.

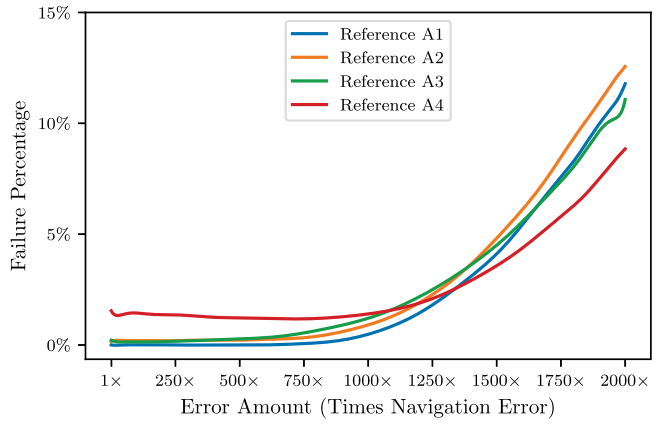



**Fig. 17.** Failure rates for Monte Carlo simulations given the four reference trajectories with 50,000 deterministic episodes each, smoothed with polynomial fit. Performance remains mostly level until the error amount exceeds the 1000× threshold.

**Table 3**
Arrival percentages for Monte Carlo simulations given various reference trajectories with 50,000 deterministic episodes each. The agent is only trained using reference A1, and generalizes to references A2, A3, and A4 (Fig. 14). The 1000 × case corresponds to the amount of error the agent experiences during training.

| Error amount | $L_1$ to $L_2$ | | $L_2$ to $L_1$ | |
|---|---|---|---|---|
| | Reference A1 | Reference A2 | Reference A3 | Reference A4 |
| 1 × | 100% | 99.9% | 99.8% | 98.5% |
| 10 × | 100% | 99.8% | 99.8% | 98.6% |
| 100 × | 100% | 99.8% | 99.9% | 98.6% |
| 1000 × | 99.5% | 99.1% | 98.8% | 98.7% |
| 2000 × | 88.5% | 87.2% | 88.7% | 91.1% |

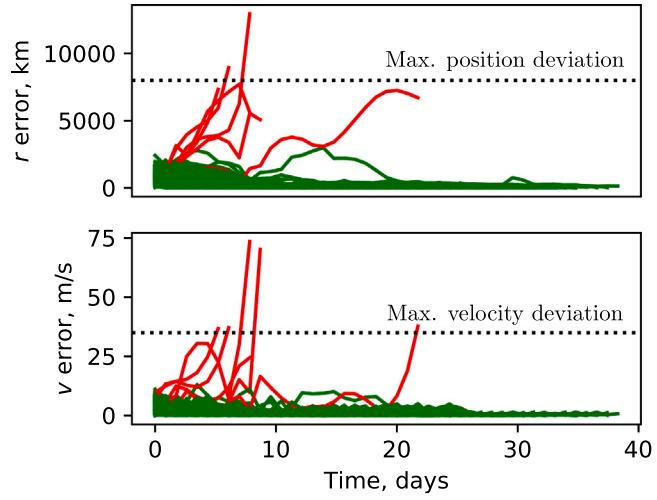



**Fig. 18.** Deviation over time for 1000 sample episodes using reference A1. The initial perturbation variance $3\sigma$ is 1000 times greater than the expected navigation error. Green denotes episodes that reach the success criteria, whereas red signifies trajectories that cross the maximum deviation threshold in position or velocity, or do not reach arrival conditions within the maximum number of time steps (87 days). In the trials, 994/1000 episodes are deemed successful for this sample batch of initial conditions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The arrival percentages for references A2, A3, and A4 further demonstrate the agent's ability to generalize experience from the training reference (A1) to the other three geometries. For reference A2, failure rates are consistently $\approx 0.2\%$ less than the A1 test case. However, given the large amount of error, the agent still arrives 99.1% of the time in the 1000× case. This high-performance level demonstrates the neural network controller's ability to perform well despite significant deviations. For reference A3, where the spacecraft does not approach the Moon, the agent performs exceedingly well, reaching the arrival criteria at least 98.8% of the time for error levels up to 1000×. Furthermore, when the agent is required to fly nearby the Moon along reference A4, it is still successful in more than 98.5% of the scenarios for error levels up to 1000×. In consistently satisfying the arrival criteria for this new scenario, the agent demonstrates a remarkable ability to generalize to new geometries. These results could potentially be improved by training the existing agent with the new transfers, i.e., employing transfer learning with the controller, but this possibility was not investigated.

Table 3 also demonstrates the controller's robustness to perturbation levels. Despite the three orders of magnitude difference between the 1× and 1000× error distributions, results remain within 1% for each reference. During training, the agent experiences 1000× error levels and, hence, learns to overcome them. When error is further increased, performance degrades. However, for values up to those used in training, performance only slightly decreases for 3 out of the 4 references, and does not degrade for Reference A4.

## 5. Training variation

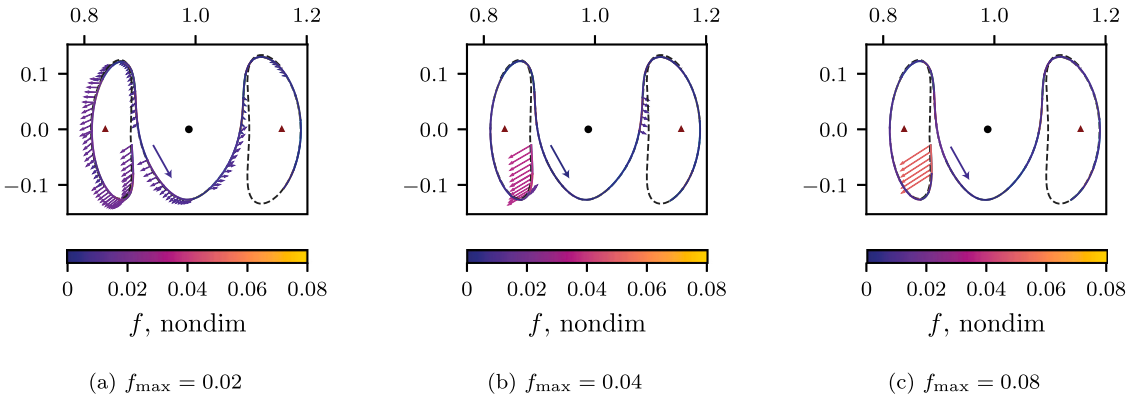
Additional transfer scenarios are examined to observe the learning framework's flexibility. Section 4 details the training process for an agent on a single reference trajectory, with other transfer scenarios introduced to evaluate generalization. By analyzing a more diverse set of agents and training references, insight is gained into the flexibility of the learning framework itself. In particular, all tuning parameters are heuristically determined to optimize the performance of the agent in Section 4. Applying the same set of parameters to different problems demonstrates the flexibility of the learning process.

### 5.1. Engine capability

To analyze the training framework's applicability to more diverse low-thrust spacecraft, two additional thrust levels are examined. As depicted in Table 1, a spacecraft with lower thrust, $f_{\max} = 0.02$ nondim, possesses thrusting capability between Hayabusa and Dawn, and a spacecraft with higher thrust, $f_{\max} = 0.08$ nondim, corresponds to a spacecraft with slightly more propulsion capability than Deep Space 1. The algorithm is easily applied to each new scenario and produces a controller that effectively commands the given engine.

Given the reference trajectory A1, Fig. 7(b), training is conducted identically to the sample agent in Section 4, with new maximum thrust levels applied. Upon completing 150,000 episodes of training, two new controllers are produced for $f_{\max} = 0.02$ and $f_{\max} = 0.08$, nondim. When overcoming simulated deviations, the resulting solutions vary based on the specific spacecraft engine characteristics. To illustrate the impact an engine exerts on the corresponding solution, a single sample perturbation is considered. The three agents, one at each thrust level, are tasked with overcoming the sample perturbation, with corresponding solutions plotted in Fig. 19. While all three episodes are deemed successful, the solutions clearly vary in the amount of initial thrusting required to return to the reference trajectory. The smallest engine, Fig. 19(a), requires many segments at nearly maximum thrust. With more propulsive capability, the middle engine, Fig. 19(b), recovers with only two large initial thrusting arcs. Finally, the largest engine employs a single significant thrusting segment with magnitude 65% of $f_{\max} = 0.08$.

Newly trained agents with $f_{\max} = 0.02$ and $f_{\max} = 0.08$ are simulated given various amounts of initial error to analyze maximum thrust and the subsequent performance of the agent. A spacecraft with more propulsive capability is expected to overcome larger deviations and, conversely, a spacecraft with less thrust will overcome smaller perturbations. The result of 10,000 error combinations with




(a) $f_{\max} = 0.02$ (b) $f_{\max} = 0.04$ (c) $f_{\max} = 0.08$

**Fig. 19.** Control history generated by three separate agents for spacecraft with various propulsive capability. As thrust increases, less time is needed to recover from the initial perturbation.

5000 episodes are each plotted in Fig. 20. Compared to the previous sample spacecraft, Fig. 10, the yellow region indicating nearly 100% efficacy extends further for the engine with a higher thrust level 20(b), and is more narrow for the less capable engine 20(a). While success varies substantially for very large error magnitudes, all three agents are nearly 100% successful for $3\sigma_r \leq 500$ km and $3\sigma_v \leq 6$ m/s. In producing effective agents for both new thrust levels, the resulting agents demonstrate the learning framework's flexibility and indicates applicability to various engine types.

### 5.2. Geometry variation

To evaluate learning framework flexibility, additional agents are produced by employing different reference geometries in the training process. New reference trajectories are plotted in Fig. 21. Each scenario, in addition to the four previous references (A1–A4), is summarized in Table 4, where each row represents a separate agent trained on the corresponding transfer. The additional transfers cover a range of energy levels, and exhibit geometries across various regions of space. Transfers (A5–A7) possess the same Jacobi constant as the four trajectories employed in Section 4, but differ in both Time of Flight (TOF) and geometry. Reference A5 departs the lunar vicinity through the $L_2$ gateway, while A6 and A7 venture toward the Earth before returning to the destination orbit.

Transfers identified with the letter 'B' correspond to higher energy Lyapunov orbits and, conversely, 'C' indicates lower energy transfers. Table 4 categorizes each transfer by geometry relative to the ZVC. "Lunar Region" indicates that the entire transfer remains in the vicinity of the Moon, "Exterior" transfers pass through the $L_2$ portal and traverse the region of space outside the ZVC, and "Interior" signifies the transfer passes through the $L_1$ portal into the enclosed region of space near Earth. After each agent is trained along its corresponding reference, the 'arrival' metric is again leveraged to evaluate performance. Employing $3\sigma_r = 1{,}000$ km and $3\sigma_v = 10$ m/s to model the initial deviation, each agent is simulated over 2000 episodes to compute an overall arrival percentage for each scenario.

High-performing agents are produced in a wide range of scenarios. References A5 and B3 demonstrate the learning framework's ability to overcome longer TOFs and exterior geometries, while A6, A7, and B4 exhibit high performance on interior references. Agents reach at least 99% arrival on transfers covering all three levels of the Jacobi constant. With several notable exceptions in mind, i.e., C3–C5, success across most scenarios indicates that the current parameters are effective in a wide range of scenarios, but are not globally applicable to every problem.

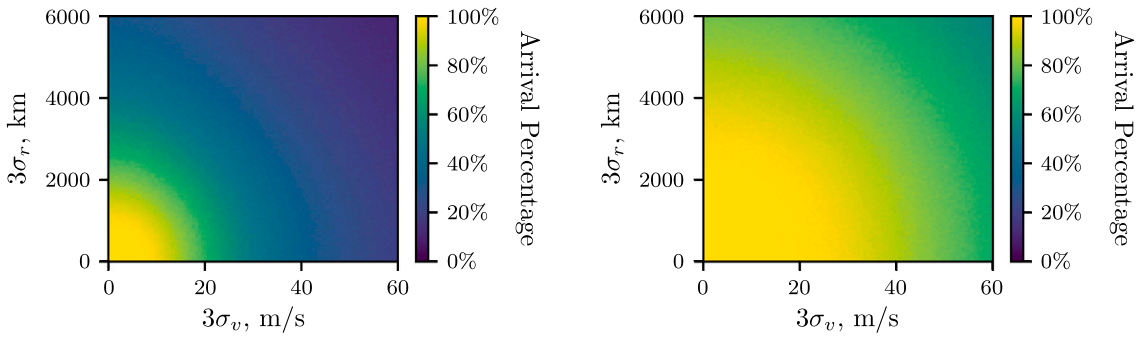
Notably sub-optimal agents are produced on three lower energy transfer scenarios (C3–C5), indicating that energy plays a key role in training parameters and agent performance. Performance degradation on these references indicates that, when changing energy levels, training framework parameters may need adjustment — in particular in response to lower energy situations. More productive agents for these scenarios are likely produced by adjusting certain parameters (e.g., time step length between successive actions, reward tuning values, PPO hyperparameters).

While the controller detailed in Section 4 demonstrates a remarkable ability in generalizing to new reference trajectories, there are several notable scenarios in which controllers are unable to generate an effective control strategy. Recall transfers are categorized by geometry in Table 4. In general, agents trained along lunar region references are unable to generalize to exterior or interior reference geometries. Conversely, since the orbits themselves are in the lunar vicinity, all agents demonstrate some ability to generalize to the lunar region, regardless of reference choice. Generalization between interior and exterior-trained agents does occur, but not always consistently. This inconsistency indicates that, if a drastic change in the reference geometry is possible, generalization performance may be increased by, during training, exposing the controller to all possible space regimes that it may encounter. One potential approach is to initialize each training episode with a randomly selected reference path, but this possibility is not investigated. The observed generalization trends do not hold for every scenario, and further research is necessary to determine precise predictions for neural network generalization behavior.

### 5.3. Energy influence on learning

Orbital energy, quantified as the value of the Jacobi constant, plays an important role in the learning framework and, therefore, in agent performance. Recall that the Jacobi constant for both the agent and the reference trajectory is included as an observation parameter in the reinforcement learning state signal, Eq. (29). This signal forms the input for the neural network controller and, hence, control policy is energy-dependent. While including these energy observations does increase agent performance, it also heightens the impact that energy exerts on the learning process. This impact is realized in Table 4, where agents fail to achieve high performance on several lower energy transfer scenarios.

To illustrate the influence of energy on the proposed guidance framework, a single sample case is introduced. In this scenario, a spacecraft has deviated from reference trajectory C1 by 3234 km and 5.3 m/s, depicted in Fig. 22, resulting in a Jacobi constant value $C = 3.172386$. This energy level possesses a forbidden region such that the $L_2$ Lyapunov orbit is no longer accessible. This closure of the $L_2$ portal is evident in Fig. 22, where the arrival portion along the reference trajectory violates the computed forbidden region depicted in gray. In returning to this reference, the agent must determine a thrust




(a) Lower thrust ($f_{\max} = 0.02$). (b) Higher thrust ($f_{\max} = 0.08$).

**Fig. 20.** Arrival percentage along the training reference (A1) given different maximum thrust capability compared to the sample spacecraft in Fig. 10. As expected, greater thrusting capability overcomes higher levels of initial error.

**Table 4**
Reference trajectories used in training. Arrival percentage computed using the training variance ($3\sigma_r = 1000$ km, $3\sigma_v = 10$ m/s).

| | Label | Figure No. | TOF (days) | Lunar region | Exterior | Interior | Arrival Pct. |
|---|---|---|---|---|---|---|---|
| | A1 | 7(b) | 43.49 | ✓ | – | – | 99.7% |
| | A2 | 14(a) | 43.49 | ✓ | – | – | 99.6% |
| | A3 | 14(b) | 39.14 | ✓ | – | – | 99.9% |
| **$C = 3.124102$** Section 4 Energy | A4 | 14(c) | 39.14 | ✓ | – | – | 99.5% |
| | A5 | 21(A5) | 119.59 | – | ✓ | – | 99.6% |
| | A6 | 21(A6) | 69.58 | – | – | ✓ | 99.1% |
| | A7 | 21(A7) | 130.46 | – | – | ✓ | 97.2% |
| | B1 | 21(B1) | 69.58 | ✓ | – | – | 98.5% |
| **$C = 3.02$** high energy | B2 | 21(B2) | 73.93 | ✓ | – | – | 94.1% |
| | B3 | 21(B3) | 121.76 | – | ✓ | – | 98.3% |
| | B4 | 21(B4) | 95.67 | – | – | ✓ | 94.2% |
| | C1 | 21(C1) | 52.18 | ✓ | – | – | 99.1% |
| | C2 | 21(C2) | 78.28 | ✓ | – | – | 91.9% |
| **$C = 3.168$** low energy | C3 | 21(C3) | 73.93 | ✓ | – | – | 78.3% |
| | C4 | 21(C4) | 132.64 | – | – | ✓ | 57.8% |
| | C5 | 21(C5) | 158.73 | – | – | ✓ | 55.2% |

history that adjusts energy such that the desired motion becomes accessible. The agent successfully recovers via the control history plotted in Fig. 23(a). The Jacobi constant over time for this solution, as depicted in Fig. 23(b), illustrates the relationship between thrust history and energy. The large initial thrusting segments rapidly increase the orbital energy, which opens the $L_2$ portal, and allows the spacecraft to return to the original reference, after which the energy remains close to the target Jacobi constant value.

Energy also impacts the agent's ability to generalize to new reference motions. For example, slightly decreasing the energy of transfer scenario A1, increasing Jacobi constant from $C = 3.124102$ to $C = 3.13$, produces a new transfer scenario depicted in Fig. 24(a). An agent is trained using reference A1, and evaluated on the newly computed reference trajectory and destination orbit. While the transfer exhibits similar geometry in configuration space to reference A1, the controller is not able to consistently execute this lower energy transfer. A sample failure case is plotted in Fig. 24(b). This limitation indicates that the agent learns the dynamics associated with a particular energy level. If energy changes are likely for a specific scenario, then the training episodes must incorporate variations in energy, or eliminate the Jacobi constant as an optional additional observation.

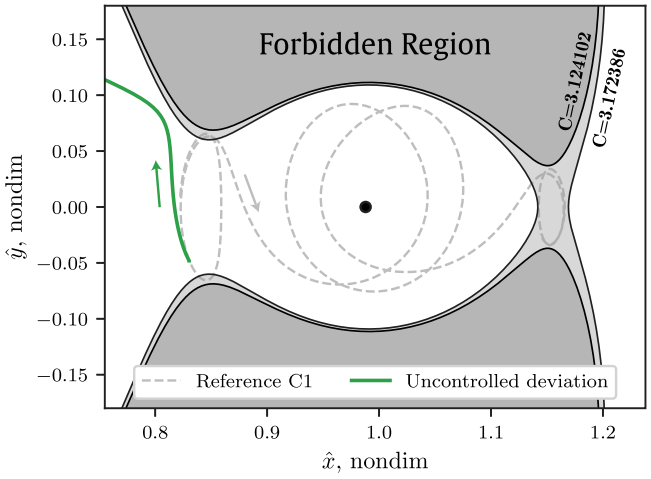
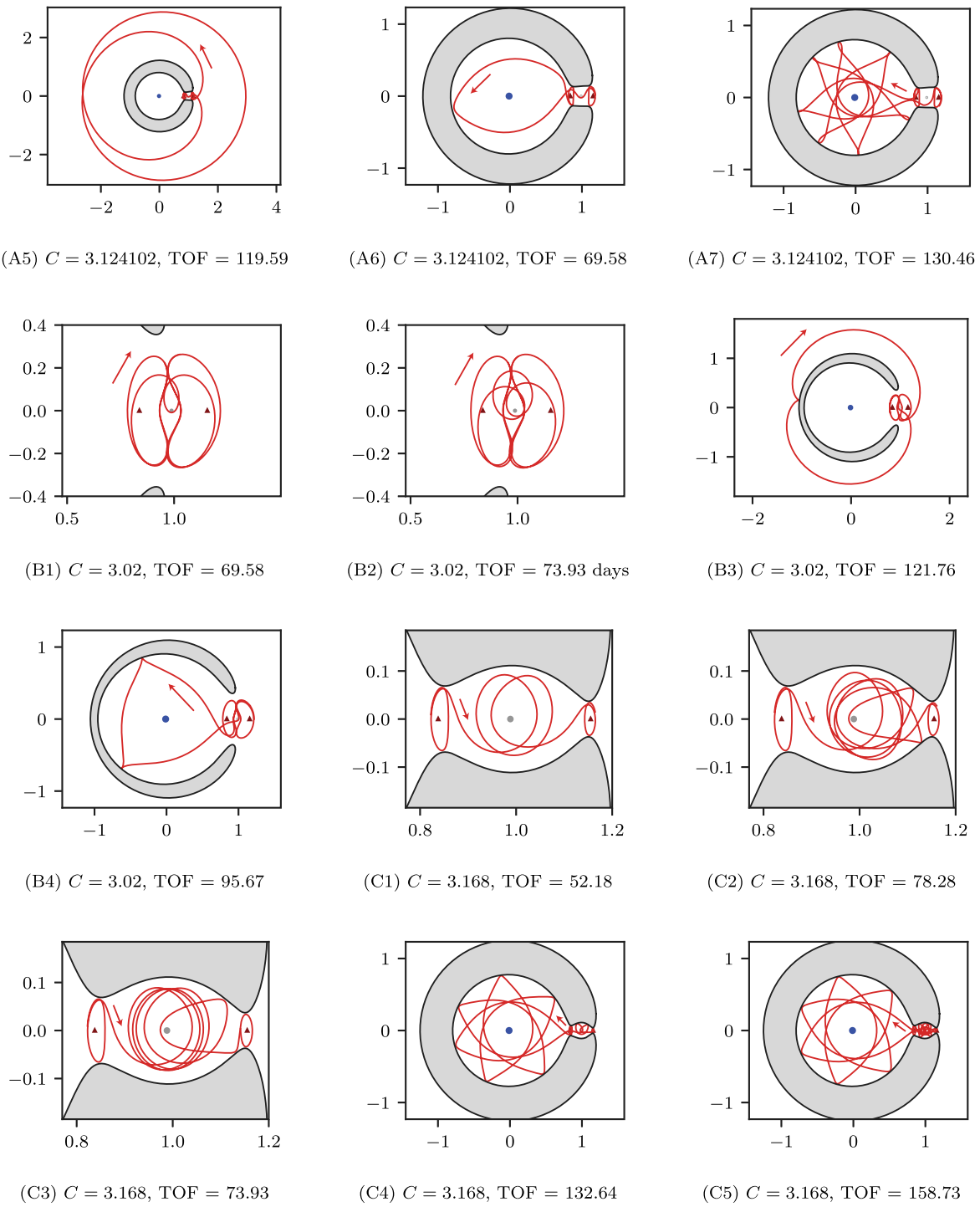



**Fig. 22.** Initial deviation at $C = 3.172386 > C_{L_2}$, closing the $L_2$ portal and restricting access to the target $L_2$ Lyapunov orbit.

## 6. Algorithm limitations

Utilizing PPO to generate a controller for path guidance offers many attractive benefits, but with several notable limitations. In particular, the stochasticity of the training process produces a nonzero probability of the agent 'getting stuck' in a local basin of attraction and failing to converge to a better policy. Hence, multiple identically configured agents can converge to drastically different policies depending on their exploration of the action space. This inconsistency in training causes difficulty with reproducibility because many agents must be simulated to discover one that achieves the desired result. The training process is computationally demanding, thus, training a large number of agents



(A5) $C = 3.124102$, TOF $= 119.59$ (A6) $C = 3.124102$, TOF $= 69.58$ (A7) $C = 3.124102$, TOF $= 130.46$

(B1) $C = 3.02$, TOF $= 69.58$ (B2) $C = 3.02$, TOF $= 73.93$ days (B3) $C = 3.02$, TOF $= 121.76$

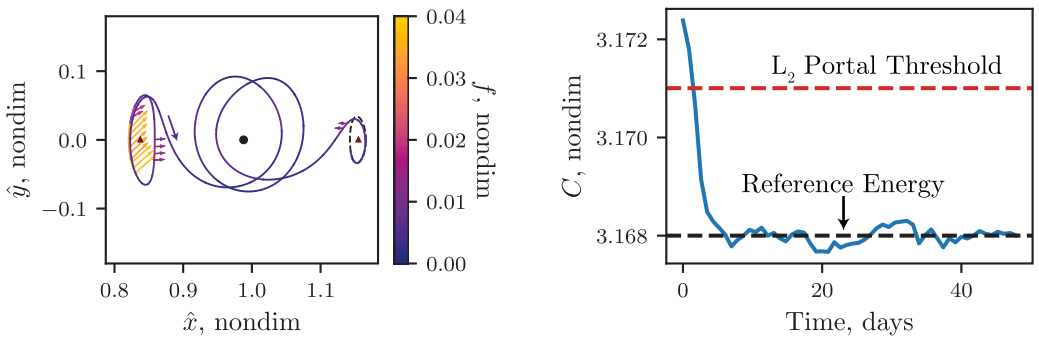
(B4) $C = 3.02$, TOF $= 95.67$ (C1) $C = 3.168$, TOF $= 52.18$ (C2) $C = 3.168$, TOF $= 78.28$

(C3) $C = 3.168$, TOF $= 73.93$ (C4) $C = 3.168$, TOF $= 132.64$ (C5) $C = 3.168$, TOF $= 158.73$

**Fig. 21.** Additional transfers employed in training ($\hat{x}$-$\hat{y}$, nondim). TOF listed in days. Blue and gray circles represent the Earth and the Moon, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

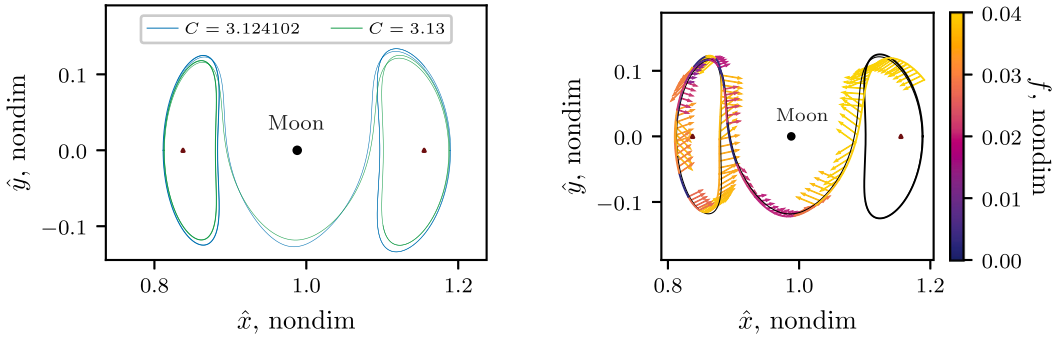
is challenging. This investigation focuses on this limitation by leveraging super-computing resources, via MIT Supercloud [69], to train many agents simultaneously. The total training time is approximately four hours per agent on an Intel Xeon Gold 6248 CPU at 2.50 GHz. Conducting episodes in parallel for a single agent may slightly decrease training time, however the on-policy assumption of PPO likely inhibits a significant decrease in training time from parallelization. Once trained, each agent is tested in a set of deterministic episodes, and the accrued reward and arrival percentages from these trials allow the extraction of desirable agents. For Section 4 scenario, approximately 26% of trained agents perform well on the training reference A1 (at least 99% arrival), though only 29% of those consistently generalize to references A2, A3, and A4 (at least 90% arrival). Hence, training many separate agents simultaneously is useful, in particular, for achieving generalization across multiple reference paths.

An additional challenge with any neural network-based guidance scheme is the 'black box' nature of the algorithm. There is no clear methodology for analytically understanding the mapping between inputs and outputs. Hence, when a failure occurs, the cause is typically diagnosed qualitatively. While existing guidance schemes can also be unpredictable, e.g., it is difficult to know *a-priori* if a targeter will converge or not, the problem is significantly more complicated with




(a) Control history (b) Jacobi constant over time

**Fig. 23.** Thrust direction and magnitude for an agent recovering from the perturbation depicted in Fig. 22 to Reference C1, Fig. 21. The trajectory is depicted in configuration space (a) and energy over time (b).




(a) Transfers at two energy levels (b) Sample episode

**Fig. 24.** Failure to generalize to a new reference trajectory at a slightly different energy level. Agent is trained at $C = 3.124102$, but is unable to complete a similar transfer geometry at $C = 3.13$.

over 10,000 trainable neural network parameters. Without analytical methods available, most studies rely on Monte Carlo analysis to evaluate algorithm performance. Such large-scale simulations are necessary to address onboard safety concerns.

Several avenues for future research are available to mitigate algorithm limitations and to offer potential improvements to the proposed training framework. While this research effort employs PPO to facilitate training, several recent investigations demonstrate alternative approaches to be effective in related problems. In particular, reinforcement meta-learning enables rapid controller adaptability by incorporating recurrent nodes into the neural network structure, and is effective in small body proximity operations [15], maneuvering target interception [29], and landing problems [70,71]. Incorporating meta-learning into the current approach may improve controller robustness to environmental and initial condition variations, e.g., multiple reference trajectories, mission scenarios, energy levels, or initial mass variations. Furthermore, promising new off-policy RL methods, such as Twin Delayed Deep Deterministic Policy Gradient (TD3) [53] and Soft Actor Critic (SAC) [54], may enable future extensions of the proposed reinforcement learning approach. Finally, additional developments are necessary to apply this analysis to a more realistic scenario in a higher-fidelity simulation. In particular, additional forces and spatial terms in the dynamical model, hardware and safety constraints, and precise error modeling are necessary prior steps in applying the proposed techniques to a practical mission scenario.

## 7. Conclusions

Computationally efficient onboard guidance is challenging in nonlinear dynamical regimes. The proposed guidance framework addresses the challenges associated with automation given limited computational resources by recasting the problem from a machine learning perspective. A controller trained via reinforcement learning overcomes perturbations and nonlinearity to autonomously compute a control history for a low-thrust spacecraft. In sample simulations, assuming perturbations three orders of magnitude greater than potential navigation errors, the controller successfully completes the given transfer in more than 99% of cases. Furthermore, sample applications demonstrate remarkable controller reconfigurability and generalization. An RL-trained controller adapts to nearby geometries not encountered during training. The relationship between performance on nearby reference paths and orbital energy suggests that energy observations and variation are important during training. This generalization ability indicates that the neural network controller continues performing in real-time despite shifts in mission objectives.

The demonstrated neural network controller offers computationally efficient closed-loop guidance in a multi-body regime, both as a standalone process, or as a hybrid approach in conjunction with an onboard targeting scheme. Many current guidance methodologies necessitate access to a high-performance computer and would not be suitable for a flight computer. Conversely, many onboard approaches are computationally efficient, but not able to take advantage of ground-based supercomputing resources. With RL, the computationally intensive training

**Table A.5**
Suggested parameter values for PPO training and CR3BP RL environment configuration.

| Variable name | Symbol | Value |
|---|---|---|
| Discount factor | $\gamma$ | 0.88 |
| Number of optimization epochs (actor) | | 20/batch |
| Number of optimization epochs (critic) | | 10/batch |
| Batch frequency | | 20 episodes |
| Reward steepness | $\lambda$ | 340 |
| Reward scaling gradient | $\xi$ | 1 |
| Reward divergence penalty | $p$ | −4 |
| Actor learning rate | | 0.00011 |
| Critic learning rate | | 0.00204 |
| KL Divergence target | $d_{\text{targ}}$ | 0.003 |
| Number of training episodes | | 150,000 |

process leverages ground-based high-performance computing, while producing a computationally efficient closed-loop controller. Furthermore, reinforcement learning approaches all separate the agent from the environment, which makes learning schemes applicable to multiple spacecraft and mission scenarios. The success of the controller across many reference geometries and thrust levels demonstrates the flexibility of the underlying learning approach. Transfers between libration point orbits support the stated conclusions and suggest the concept that RL-trained controllers are effective in challenging dynamical regions of space.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

The authors thank the Purdue University School of Aeronautics and Astronautics and the Massachusetts Institute of Technology Department of Aeronautics and Astronautics for facilities and support in conducting this research. Additionally, the authors wish to thank members of the Multibody Dynamics Research Group at Purdue, as well as members of the Space Systems Lab at MIT for invaluable discussions. Special thanks to Dave Folta (NASA Goddard Space Flight Center) for helpful feedback, and to our two anonymous reviewers for their careful reading and valuable suggestions. The authors acknowledge the MIT Supercloud and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the research results reported within this paper. This work was supported by NASA Space Technology Research Fellowships, NASA Grant 80NSSC19K1175 and NASA Grant 80NSSC18K1141.

## Appendix. Reinforcement learning parameter selection

Parameter selection and tuning is an important aspect in both PPO learning and RL environment design. Well-performing agents may be produced with different combinations of parameters. Values employed in this research are summarized in Table A.5, with additional suggested values included to indicate cases where desirable behavior is observed given different parameter values. Furthermore, the specific configurations of the employed neural networks are listed in Table A.6. These networks are implemented using TensorFlow [72].

**Table A.6**
Configuration of actor and critic neural networks employed in this investigation.

| Layer name | Symbol | Actor | | Critic | |
|---|---|---|---|---|---|
| | | Size | Activation function | Size | Activation function |
| Input layer | $I$ | 11 | tanh | 11 | tanh |
| Hidden 1 | $H^1$ | 120 | tanh | 120 | tanh |
| Hidden 2 | $H^2$ | 60 | tanh | 24 | tanh |
| Hidden 3 | $H^3$ | 30 | tanh | 5 | tanh |
| Output | $O$ | 3 | tanh | 1 | linear |

## References

[1] J. Hart, E. King, P. Miotto, S. Lim, Orion gn & c architecture for increased spacecraft automation and autonomy capabilities, in: AIAA Guidance, Navigation and Control Conference and Exhibit, AIAA, Honolulu, Hawaii, 2008, pp. 1–26.

[2] K.L. Wagstaff, G. Doran, A. Davies, S. Anwar, S. Chakraborty, M. Cameron, I. Daubar, C. Phillips, Enabling onboard detection of events of scientific interest for the europa clipper spacecraft, in: 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Association for Computing Machinery, Anchorage, Alaska, 2019, pp. 2191—-2201, http://dx.doi.org/10.1145/3292500.3330656.

[3] N. Bosanac, A.D. Cox, K.C. Howell, D.C. Folta, Trajectory design for a cislunar cubesat leveraging dynamical systems techniques: The lunar icecube mission, Acta Astronaut. 144 (2018) 283–296.

[4] C. Hardgrove, J. Bell, R. Starr, T. Colaprete, M. Robinson, D. Drake, I. Labzin, G. West, E. Johnson, J. Christian, A. Genova, D. Dunham, B. Williams, D. Nelson, A. Babuscia, P. Scowen, K. Cheung, M. Klesh, H. Kerner, A. Deran, R. Amzler, Z. Burnham, J. Lightholder, P. Wren, A. Godber, M. Beasley, The lunar polar hydrogen mapper (lunah-map) cubesat mission, in: 47th Lunar and Planetary Science Conference, Lunar and Planetary Institute, The Woodlands, Texas, 2016, pp. 1–2.

[5] W. Hart, G.M. Brown, S.M. Collins, M. De Soria-Santacruz Pich, P. Fieseler, D. Goebel, D. Marsh, D.Y. Oh, S. Snyder, N. Warner, G. Whiffen, L.T. Elkins-Tanton, J.F. Bell III, D.J. Lawrence, P. Lord, Z. Pirkl, Overview of the spacecraft design for the psyche mission concept, in: 2018 IEEE Aerospace Conference, IEEE, Big Sky, Montana, 2018, pp. 1–20.

[6] D. Irimies, D. Manzella, T. Ferlin, Summary of gateway power and propulsion element (ppe) studies, in: 2019 IEEE Aerospace Conference, IEEE, Big Sky, Montana, 2019, pp. 1–6.

[7] M.A. Vavrina, J.A. Englander, S.M. Phillips, K.M. Hughes, Global, multi-objective trajectory optimization with parametric spreading, in: AAS/AIAA Astrodynamics Specialist Conference, American Astronautical Society, Stevenson, Washington, 2017, pp. 1–20.

[8] C. Ocampo, Finite burn maneuver modeling for a generalized spacecraft trajectory design and optimization system, Ann. New York Acad. Sci. 1017 (2004) 210–233.

[9] B.G. Marchand, M.W. Weeks, C.W. Smith, S. Scarritt, Onboard autonomous targeting for the trans-earth phase of orion, J. Guid. Control Dyn. 33 (2010) 943–956, doi:10.2514/1.42384, https://doi.org/10.2514/1.42384.

[10] S.K. Scarritt, B.G. Marchand, A.J. Brown, W.H. Tracy, M.W. Weeks, Finite-burn linear targeting algorithm for autonomous path planning and guidance, J. Guid. Control Dyn. 35 (2012) 1605–1615.

[11] A.F. Haapala, K.C. Howell, A framework for constructing transfers linking periodic libration point orbits in the spatial circular restricted three-body problem, Int. J. Bifurcations Chaos 26 (2016).

[12] B. Marchand, S. Scarritt, T. Pavlak, K. Howell, A dynamical approach to precision entry in multi-body regimes: Dispersion manifolds, Acta Astronaut. 89 (2013) 107–120.

[13] J.D. Yencharis, R.F. Wiley, R.S. Davis, Q.A. Holmes, K.T. Zeiler, Apollo Experience Report: Development of Guidance Targeting Techniques for the Command Module and Launch Vehicle, Techreport, National Aeronautics and Space Administration, 1972.

[14] B. Gaudet, R. Linares, R. Furfaro, Deep reinforcement learning for six degree-of-freedom planetary landing, Adv. Space Res. 65 (2020) 1723–1741.

[15] B. Gaudet, R. Linares, R. Furfaro, Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations, Acta Astronaut. 171 (2020) 1–13.

[16] A. Rubinsztejn, R. Sood, F.E. Laipert, Neural network optimal control in astrodynamics: Application to the missed thrust problem, Acta Astronaut. 176 (2020) 192–203.

[17] T.A. Estlin, B.J. Bornstein, D.M. Gaines, R.C. Anderson, D.R. Thompson, M. Burl, R. Castaño, M. Judd, Aegis automated science targeting for the mer opportunity rover, ACM Trans. Intell. Syst. Technol. (TIST) 3 (2012) 1–19.

[18] R. Francis, T. Estlin, G. Doran, S. Johnstone, D. Gaines, V. Verma, M. Burl, J. Frydenvang, S. Montano, R. Wiens, S. Schaffer, O. Gasnault, L. Deflores, D. Blaney, B. Bornstein, Aegis autonomous targeting for chemcam on mars science laboratory: Deployment and results of initial science team use, Science Robotics 2 (2017).

[19] S. Higa, Y. Iwashita, K. Otsu, M. Ono, O. Lamarre, A. Didier, M. Hoffmann, Vision-based estimation of driving energy for planetary rovers using deep learning and terramechanics, IEEE Robot. Autom. Lett. 4 (2019) 3876–3883.

[20] B. Rothrock, J. Papon, R. Kennedy, M. Ono, M. Heverly, C. Cunningham, Spoc: Deep learning-based terrain classification for mars rover missions, in: AIAA Space and Astronautics Forum and Exposition, SPACE 2016, American Institute of Aeronautics and Astronautics Inc, AIAA, 2016, pp. 1–12.

[21] B. Dachwald, Evolutionary neurocontrol: A smart method for global optimization of low-thrust trajectories, in: AIAA/AAS Astrodynamics Specialist Conference and Exhibit, Providence, Rhode Island, 2004, pp. 1–16.

[22] S. De Smet, D.J. Scheeres, Identifying heteroclinic connections using artificial neural networks, Acta Astronaut. 161 (2019) 192–199.

[23] N.L.O. Parrish, Low Thrust Trajectory Optimization InCislunar and Translunar Space (Ph.D. thesis), University of Colorado Boulder, 2018.

[24] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S.M.A. Eslami, M.A. Riedmiller, D. Silver, Emergence of locomotion behaviours in rich environments, 2017, CoRR abs/1707.02286 arXiv:1707.02286.

[25] 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, D. Hassabis, Mastering the game of go without human knowledge, Nature 550 (2017) Article.

[26] R. Furfaro, A. Scorsoglio, R. Linares, M. Massari, Adaptive generalized zem-zev feedback guidance for planetary landing via a deep reinforcement learning approach, Acta Astronaut. 171 (2020) 156–171.

[27] B. Gaudet, R. Linares, R. Furfaro, Six degree-of-freedom hovering over an asteroid with unknown environmental dynamics via reinforcement learning, in: 20th AIAA Scitech Forum, AIAA, Orlando, Florida, 2020, pp. 1–15, http://dx.doi.org/10.2514/6.2020-1910, https://arc.aiaa.org/doi/abs/10.2514/6.2020-1910.

[28] J. Broida, R. Linares, Spacecraft rendezvous guidance in cluttered environments via reinforcement learning, in: 29th AAS/AIAA Space Flight Mechanics Meeting, American Astronautical Society, Ka'anapali, Hawaii, 2019, pp. 1–15.

[29] B. Gaudet, R. Furfaro, R. Linares, Reinforcement learning for angle-only intercept guidance of maneuvering targets, Aerosp. Sci. Technol. 99 (2020).

[30] D. Guzzetti, Reinforcement learning and topology of orbit manifolds for station-keeping of unstable symmetric periodic orbits, in: AAS/AIAA Astrodynamics Specialist Conference, American Astronautical Society, Portland, Maine, 2019, pp. 1–20.

[31] J.A. Reiter, D.B. Spencer, Augmenting spacecraft maneuver strategy optimization for detection avoidance with competitive coevolution, in: 20th AIAA Scitech Forum, AIAA, Orlando, Florida, 2020, pp. 1–11, http://dx.doi.org/10.2514/6.2020-1910, https://arc.aiaa.org/doi/abs/10.2514/6.2020-1910.

[32] A. Das-Stuart, K.C. Howell, D.C. Folta, Rapid trajectory design in complex environments enabled by reinforcement learning and graph search strategies, Acta Astronaut. 171 (2020) 172–195, http://dx.doi.org/10.1016/j.actaastro.2019.04.037.

[33] D. Miller, R. Linares, Low-thrust optimal control via reinforcement learning, in: 29th AAS/AIAA Space Flight Mechanics Meeting, American Astronautical Society, Ka'anapali, Hawaii, 2019, pp. 1–18.

[34] C.J. Sullivan, N. Bosanac, Using reinforcement learning to design a low-thrust approach into a periodic orbit in a multi-body system, in: 20th AIAA Scitech Forum, AIAA, Orlando, Florida, 2020, pp. 1–19, http://dx.doi.org/10.2514/6.2020-1910, https://arc.aiaa.org/doi/abs/10.2514/6.2020-1910.

[35] A.D. Cox, A Dynamical Systems Perspective for Preliminary Low-Thrust Trajectory Design in Multi-Body Regimes (Ph.D. thesis), Purdue University, 2020.

[36] M.D. Rayman, P. Varghese, D.H. Lehman, L.L. Livesay, Results from the deep space 1 technology validation mission, Acta Astronaut. 47 (2000) 475–487.

[37] C. Russell, C. Raymond, The Dawn Mission To Minor Planets 4 Vesta and 1 Ceres, 1st ed. ed., Springer, 2012.

[38] A. Cox, K. Howell, D. Folta, Dynamical structures in a low-thrust, multi-body model with applications to trajectory design, Celestial Mech. Dynam. Astronom. 131 (2019) 1–34.

[39] Busek BIT-3 RF Ion Thruster, BIT-3 RF Ion Thruster, Busek Co. Inc., 2019.

[40] H. Kuninaka, K. Nishiyama, Y.S.I. Funaki, H. Koizumi, Hayabusa asteroid explorer powered by ion engines on the way to earth, in: 31st International Electric Propulsion Conference, Ann Arbor, Michigan, 2009, pp. 1–6.

[41] K. Nishiyama, S. Hosoda, K. Ueno, R. Tsukizaki, H. Kuninaka, Development and testing of the hayabusa2 ion engine system, Trans. Japan Soc. Aeronaut. Space Sci. 14 (2016) 131–140.

[42] J.S. Snyder, G. Lenguito, J. Frieman, T. Haag, J. Mackey, The effects of background pressure on spt-140 hall thruster performance, in: 53rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference, AIAA, Cincinnati, Ohio, 2018, pp. 1–17.

[43] A. Das, Artificial Intelligence Aided Rapid Trajectory Design in Complex Dynamical Environments (Ph.D. thesis), Purdue University, 2019.

[44] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning Data Mining, Inference, and Prediction, second ed., in: Springer Series in Statistics, Springer, 2009.

[45] S. Robinson, S. Scarritt, J.L. Goodman, Encke-beta predictor for orion burn targeting and guidance, in: 39th Annual AAS Guidance and Control Conference, AAS, Breckenridge, CO, 2016, pp. 709–731.

[46] S.S. Karri, Nasa sbir 2019 phase i solicitation: Deep neural net and neuromorphic processors for in-space autonomy and cognition, 2019, online, https://sbir.nasa.gov/printpdf/61636.

[47] G. Bersuker, M. Mason, K.L. Jones, Neuromorphic Computing: The Potential for High-Performance Processing in Space, Technical Report, The Aerospace Corporation, 2018.

[48] R.S. Sutton, A.G. Barto, Reinfrocement Learning: An Introduction, second ed., The MIT Press, 2018.

[49] J. Schulman, S. Levine, P. Moritz, M.I. Jordan, P. Abbeel, Trust region policy optimization, 2015, CoRR abs/1502.05477.

[50] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, 2015, arXiv preprint arXiv:1506.02438.

[51] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, 2016, CoRR abs/1602.01783 arXiv:1602.01783.

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

[53] S. Fujimoto, H. van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: 35th International Conference on Machine Learning (ICML), 2018, pp. 1–15, https://arxiv.org/pdf/1802.09477.pdf.

[54] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: A. Krause J. Dy (Ed.), Proceedings of the 35th International Conference on Machine Learning, Volume 80 of Proceedings of Machine Learning Research, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, pp. 1861–1870, http://proceedings.mlr.press/v80/haarnoja18b.html.

[55] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, CoRR abs/1707.06347.

[56] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (1992) 229–256, http://dx.doi.org/10.1007/BF00992696.

[57] P. Coady, Proximal policy optimization with generalized advantage estimation, 2018, https://github.com/pat-coady/trpo.

[58] R. Rojas, The Backpropagation Algorithm, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 149–182, http://dx.doi.org/10.1007/978-3-642-61068-4_7.

[59] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, CoRR abs/1412.6980. arXiv:1412.6980.

[60] D. Miller, J.A. Englander, R. Linares, Interplanetary low-thrust design using proximal policy optimization, in: AAS/AIAA Astrodynamics Specialist Conference, American Astronautical Society, Portland, Maine, 2019, pp. 1–16.

[61] C.E. York, Application of a Two-Level Targeter for Low-Thrust Spacecraft Trajectories (Master's thesis), Purdue University, 2018.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[63] M. Vaquero, J. Senent, Poincare : A multibody, multi-system trajectory design tool, in: 7th International Conference on Astrodynamics Tools and Techniques, Oberpfaffenhofen, Germany, 2018, pp. 1–12.

[64] R. Pritchett, K.C. Howell, D.C. Folta, Low-thrust trajectory design for a cislunar cubesat leveraging structures from the bicircular restricted four-body problem, in: 70th International Astronautical Congress, Washington D.C. USA, 2019, pp. 1–18.

[65] C. Forbes, M. Evans, N. Hastings, B. Peacock, Statistical Distributions, fourth ed., John Wiley & Sons, Ltd, 2010, pp. 69–73, http://dx.doi.org/10.1002/9780470627242.ch11.

[66] D. Guzzetti, E.M. Zimovan, K.C. Howell, D.C. Davis, Stationkeeping analysis for spacecraft in lunar near rectilinear halo orbits, in: 27th AAS/AIAA Space Flight Mechanics Meeting, American Astronautical Society, San Antonio, Texas, 2017, pp. 1–24.

[67] G. Bozis, Zero velocity surfaces for the general planar three-body problem, Astrophys. Space Sci. 43 (1976) 355–368.

[68] D. Davis, S. Bhatt, K. Howell, J.-W. Jang, R. Whitley, F. Clark, D. Guzzetti, E. Zimovan, G. Barton, Orbit maintenance and navigation of human spacecraft at cislunar near rectilinear halo orbits, in: 27th AAS/AIAA Space Flight Mechanics Meeting, American Astronautical Society, San Antonio, Texas, 2017, pp. 1–20.

[69] A. Reuther, J. Kepner, C. Byun, S. Samsi, W. Arcand, D. Bestor, B. Bergeron, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, L. Milechin, J. Mullen, A. Prout, A. Rosa, C. Yee, P. Michaleas, Interactive supercomputing on 40, 000 cores for machine learning and data analysis, in: 2018 IEEE High Performance Extreme Computing Conference (HPEC), 2018, pp. 1–6.

[70] B. Gaudet, R. Linares, R. Furfaro, Adaptive guidance and integrated navigation with reinforcement meta-learning, Acta Astronaut. 169 (2020) 180–190.

[71] A. Scorsoglio, R. Furfaro, R. Linares, B. Gaudet, Image-based deep reinforcement learning for autonomous lunar landing, in: 20th AIAA Scitech Forum, AIAA, Orlando, Florida, 2020, pp. 1–16, http://dx.doi.org/10.2514/6.2020-1910, https://arc.aiaa.org/doi/abs/10.2514/6.2020-1910.

[72] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous systems, 2015, https://www.tensorflow.org/, software available from tensorflow.org.