Guidance for Closed-Loop Transfers using Reinforcement Learning with Application to Libration Point Orbits

Nicholas B. LaFarge*
Purdue University, West Lafayette, IN 47907-2045

Daniel Miller†
Massachusetts Institute of Technology, Cambridge, MA 02138-4301

Kathleen C. Howell‡
Purdue University, West Lafayette, IN 47907-2045

Richard Linares§
Massachusetts Institute of Technology, Cambridge, MA 02138-4301

While human presence in cislunar space continues to expand, so too does the demand for 'lightweight' automated on-board processes. In nonlinear dynamical environments, computationally efficient guidance strategies are challenging. Many traditional approaches rely on either simplifying assumptions in the dynamical model or abundant computational resources. The proposed controller employs the use of the nonlinear equations of motion without imposing a heavy workload on a flight computer. The guidance framework is nevertheless able to leverage high-performance computing by separating the training from the resulting controller. Practical examples demonstrate the flexibility of a reinforcement learning approach, and suggest extendability to higher-fidelity domains.

I. Nomenclature

$\boldsymbol{a_t}$	=	action selected by reinforcement learning agent at time t
a_{lt}	=	nondimensional acceleration of a low-thrust engine
$\hat{A}_t$	=	advantage function estimate at time t
b	=	reward bonus for reaching arrival condition in a target orbit
B	=	barycenter between P_1 and P_2 in the CR3BP
C	=	Jacobi constant associated with a particular energy level in the CR3BP
d	=	Kullback–Leibler divergence
δ	=	maximum allowable Kullback-Leibler divergence
$\boldsymbol{\delta\rho}$	=	relative position and velocity from a spacecraft to its nearest neighbor on a reference trajectory
ϵ	=	PPO clipping factor
η	=	scaling term to increase reward along a given reference path
f	=	nondimensional spacecraft thrust magnitude
g	=	gradient of reinforcement learning policy function
γ	=	discount factor to prioritize immediate vs. future rewards
H	=	hidden layer of neural network
I_{sp}	=	Specific impulse of a low-thrust engine
k	=	L2-norm of the relative position and velocity from a spacecraft to a reference trajectory
λ	=	scaling factor to adjust the gradient of reward increase near a reference trajectory
l^*	=	distance between the primary bodies in the CR3BP
L	=	loss function for reinforcement learning objectives

*Graduate Student, School of Aeronautics and Astronautics, nlafarge@purdue.edu, Student Member AIAA.
†Graduate Student, Department of Aeronautics and Astronautics, dmmiller@mit.edu, Student Member AIAA.
‡Hsu Lo Distinguished Professor of Aeronautics and Astronautics, School of Aeronautics and Astronautics, howell@purdue.edu, AIAA Fellow.
§Charles Stark Draper Assistant Professor, Department of Aeronautics and Astronautics, linaresr@mit.edu, Senior Member AIAA

m = nondimensional ratio of the mass of the spacecraft to its initial mass
M_3 = dimensional mass of the third body in the CR3BP
μ = ratio between the larger and smaller primary body masses in the CR3BP
p_i = penalty imposed for impacting the primary or secondary body
p_d = penalty imposed for deviating from the reference trajectory
P_3 = third body in the CR3BP, frequently used to represent a spacecraft
π = reinforcement learning agent policy
$\boldsymbol{q_t}$ = dynamical state at time t containing position, velocity and mass information
r = reinforcement learning environmental reward
$\boldsymbol{r_3}$ = vector locating the position of P_3 in the CR3BP
$\mathcal{R}^\mathrm{ref}$ = set of discrete states along a reference trajectory
$\boldsymbol{\rho}$ = vector containing position and velocity information in a rotating reference frame
$\boldsymbol{s_t}$ = state at time t
t^* = characteristic time of the CR3BP
θ = parameters for the reinforcement learning policy
$\hat{u}$ = thrust direction with respect to the CR3BP rotating frame
V = estimation of value function
ξ = scaling term to adjust the rate at which reward increases over a reference

II. Introduction

ESTABLISHING a permanent foothold in cislunar space is considered by NASA as pivotal in expanding human exploration. With the proposed Gateway and Artemis projects, NASA aims to develop the capability for an enduring human presence beyond Low Earth Orbit (LEO). This development effort involves accessing complex dynamical structures, e.g., a Near Rectilinear Halo Orbit (NRHO), that only exist in force models that incorporate the gravity of both the Earth and the Moon simultaneously [1]. Furthermore, increasingly complex spacecraft require more autonomous on-board computational capability than previous flight systems. Orion, in particular, is required to include the "capability to automatically execute GN&C functionality during all phases of flight " [2]. The increased complexity introduces a unique challenge for trajectory designers. For example, many traditional Keplerian-based guidance approaches are infeasible due to the nonlinearity in the dynamical model while, in contrast, many modern strategies rely on abundant computational resources not available on a flight computer. This investigation addresses these challenges by demonstrating Reinforcement Learning (RL), a subset of machine learning, to be a computationally 'lightweight' approach for automated closed-loop guidance in support of multi-body orbit transfers.

In recent years, RL has proven beneficial in achieving state-of-the-art performance in historically challenging domains despite significant environmental uncertainty [3]. While recent progress has addressed these dynamical challenges in the Circular Restricted Three-Body Problem (CR3BP), an additional complicating factor is the planned inclusion of solar electric propulsion options on Orion and Gateway that prohibit instantaneous maneuvers in favor of gradual velocity and energy changes over longer time intervals. Furthermore, corresponding solutions often rely on the underlying assumptions in the force model and are not easily extendable to higher-fidelity domains. Yet preliminary analysis is frequently accomplished in the CR3BP and then transitioned to an ephemeris model for a higher-fidelity solution. Conversely, using RL as a model-agnostic guidance approach is more adaptable to different domains and offers direct interaction with complex dynamical models.

While many recent advancements in trajectory design exploit improvements in computer hardware, relatively few are practical for autonomous on-board implementations given the limited computational resources. In multi-body problems, trajectory designers often generate low-cost initial guesses for transfers by leveraging dynamical systems theory and invariant manifolds [4]. Dynamical systems-based approaches have been useful in many previous applications and, when combined with differential corrections and/or optimization techniques, yield globally optimal solutions for many applications. However, this approach is computationally intensive and often requires human-in-the-loop interactions. An alternative strategy bases outcomes on trajectory design and optimization tools, e.g., Copernicus [5] and/or NASA's Evolutionary Mission Trajectory Generator (EMTG) [6]. But, these computational tools, while powerful, often involve lengthy grid search and optimization processes that prohibit rapid trajectory and control history construction. As NASA expands both a human and robotic presence in cislunar space and beyond, the limitations of these methods suggest that new approaches are necessary to address the time and computational cost of current procedures.

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. However, within the context of flight software, optimality is considered secondary to feasibility [7]; the capability to rapidly re-compute a reference path and a feasible control history in-flight is critical for autonomous guidance. By approaching on-board 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 on-board controller and, thus, leverages modern computer hardware while still producing a controller sufficiently lightweight for use in flight.

III. Reinforcement Learning Formulation

Machine learning, as a spacecraft trajectory design and optimization tool, has only recently been introduced. The applications are broadly grouped into two categories: supervised learning and reinforcement learning. Supervised learning typically involves classification or regression tasks by optimizing an artificial Neural Network (NN) to map selected inputs to known outputs. Recent advancements in the development of deep NNs (NNs with multiple hidden layers) render such approaches accessible and productive. In trajectory design, Dachwald used a shallow NN for low-thrust trajectory optimization as early as 2004 [8]. More recently, De Smet and Scheeres employed a NN to identify heteroclinic connections in the CR3BP [9]; Parrish and Scheeres then explored many nearby optimized trajectories to train a NN to generate low-thrust trajectory corrections [10]. Furfaro et al. explored supervised learning to model a fuel-optimal lunar landing control history [11]. In these supervised learning approaches, algorithms rely on a large amount of accurate training data, generated by the user, and their performance is largely dependent on the quality and quantity of this data.

Reinforcement learning can also train a neural network controller, but the training process is very different. Rather than demanding a large quantity of known representative training data, an RL agent interacts directly with its environment, and creates this training knowledge 'on-the-fly' based on a reward response. While there is tremendous advantage to removing the need for *a priori* system knowledge in training an RL agent, challenges are also inherent in this approach. In particular, an agent's performance depends on a Markovian system response and is highly dependent on the choice of the reward function, which is often unique to any given application [3]. Within RL, algorithms are frequently broadly classified as value optimization or policy optimization schemes. The former framework involves an agent that learns the 'value' of a particular state and action pair, often by a process of iteration using the Bellman equation. The value of a state is analogous to the cost-to-go function in an optimal control problem. In most modern RL implementations, this function is approximated using a NN, though it can also be tabular in problems with small and easily discretized state and action spaces. Having learned the optimal value function, the optimal policy is easily determined by selecting the action at each state with the highest value. Policy optimization, in contrast, involves learning a policy function that yields the optimal action for a given state. This alternate approach is iterative such that a policy is leveraged to compute a corresponding value function, which in turn determines improved policy and value functions. Unfortunately, value optimization methods, also denoted critic-only schemes, are computationally expensive due to a requirement that every possible state be visited. Policy optimization approaches do not require such completeness. However, most such strategies belong to a subcategory labeled policy gradient methods that suffer from slow learning due to high variances in the estimates of gradients [12].

Due to the limitations of both policy and value optimization schemes, a class of hybrid algorithms, denoted actor-critic methods, are of particular interest. These approaches combine both value and policy paradigms into one unified approach. In recent years, actor-critic strategies have proven especially effective in problems involving continuous control tasks. One such algorithm is Proximal Policy Optimization (PPO). This investigation focuses on PPO as the core RL algorithm due to its demonstrated performance in continuous domains and its relative simplicity in implementation.

Some recent advancements in RL for astrodynamics applications are notable. For path-finding, in 2017 Das-Stuart et al. leveraged Q-Learning in conjunction with accessible regions to compute initial guesses for low-thrust transfers in the CR3BP [13]. Following this effort, in 2019, Das-Stuart et al. furthered the investigation by leveraging supervised learning to influence the resulting solution geometry [14], and applied the framework to contingency planning [15]. As applied toward other related tasks, Gaudet et al. used PPO in an optimal control Martian landing guidance problem [16], Furfaro et al. took a ZEM/ZEV feedback approach to lunar lander guidance [17], Broida and Linares used RL for rendezvous guidance in a cluttered environment [18], and Guzzetti applied RL to multi-body station-keeping [19]. In 2019, Miller and Linares first employed PPO for low-thrust trajectory optimization in a multi-body regime [20]. The same year, Miller et al. further demonstrated PPO's decision-making capability for interplanetary low-thrust transfers [21], Reiter et al. employed PPO for spacecraft detection avoidance [22], and Gaudet et al. proposed an

adaptive Guidance, Navigation and Control (GN&C) framework for asteroid proximity operations using PPO [23]. This investigation builds on these approaches by demonstrating the ability of PPO to control a spacecraft in the nonlinear planar CR3BP, while also exploring the RL framework, in particular, the impact of the state and reward signals on learning performance.

A. Neural Networks

Neural networks are employed extensively in modern RL algorithms due to their demonstrated ability in approximating policy and value functions. Many traditional tabular RL approaches, such as Q-learning, rely on finely discretizing the state and action spaces, which quickly becomes impractical as the number of dimensions in the problem grows. This drawback is known as the "curse of dimensionality" [3]. Leveraging NNs allow modern algorithms to both access the continuous state and action spaces, and to easily include additional dimensions.

Neural networks employed in this investigation are composed of an input layer, three intermediate layers, whose outputs are 'hidden', and an output layer, summarized in Table 5. Each layer is composed of nodes, called neurons. In each neuron, the sum of the weighted outputs from the previous layer, combined with a bias term, are processed through a nonlinear activation function to produce outputs. Hence, the entire neural network is evaluated using a combination of matrix operations and element-wise nonlinear activation functions. The activation function employed in this investigation is hyperbolic tangent, or `tanh`. With this simplicity in evaluation, a neural network controller is computationally efficient, and can be applicable flight computers.

Despite the relative simplicity in evaluation, implementing a NN on a flight computer presents additional challenges due to "significant amounts of multiply and accumulation operations" and a "substantial amount of memory to store data" [24]. Flight hardware to address these difficulties is currently being developed. NASA is actively soliciting proposals for neuromorphic processors to enable in-space autonomy [24]. These specialized processors, inspired by the human brain [25], allow for dedicated low-power NN evaluation in space. Adoption of neuromorphic hardware into flight systems will render machine learning approaches more accessible and productive. In particular, neuromorphic computing could enable efficient autonomous control, decision making, and onboard adaptive learning [26],

A sample NN used in this investigation is depicted in Fig. 1. The input I to the network contains 11 values and is mapped to 120 neurons in the first hidden layer H^1. To achieve this, a matrix of weights $\mathcal{W}^1$ with dimensions 120 by 11 and a matrix of bias terms $\mathcal{B}^1$ with dimensions 120 by 1 are defined. With the activation function, α^1, being applied element-wise to each neuron, the calculation from the input layer to the first hidden layer is,

$$H^1_{120\times 1} = \alpha^1\left(\mathcal{W}^1_{120\times 11} I_{11\times 1} + \mathcal{B}^1_{120\times 1}\right) \tag{1}$$

An identical process is then repeated between each layer of the network until the output is reached. In the sample network shown, this final output consists of three scalar values, $[\tilde{f},\ \tilde{u}_x,\ \tilde{u}_y]$.

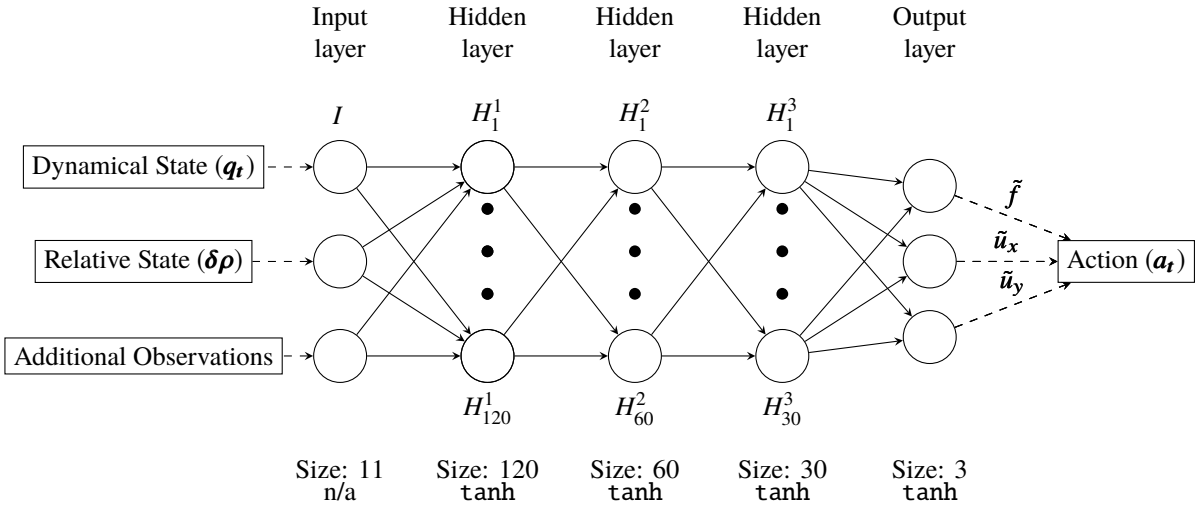


Fig. 1 Actor network employed in PPO algorithm

B. Foundational RL Concepts

In RL, an agent is a controller that maps observed variables to actions. This mapping is known the agent's policy (π). The goal of the agent's training process is to improve the policy such that the agent learns which actions are desirable. Training is a repetitive process that is composed of many thousands of episodes, or attempts at solving the desired problem. These episodes are further subdivided into tens or hundreds of time steps. At each time step, the agent must take the current observation, and compute an action. Eventually, the agent is able to converge on an optimal deterministic policy (π^*), which can then be used as a controller for the given problem.

The learning framework is designed as a Markov Decision Process (MDP) and, thus, the 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 [3]. In many practical applications, only partial information is available, and the agent instead receives a subset of all environmental data. This signal is denoted the 'observation'. The delineation between state and observation is important in partially observable systems. However, this investigation assumes a fully observable MDP and, thus, for simplification, the observation $\boldsymbol{o_t}$ can be represented as the state $\boldsymbol{s_t}$.

The specific mechanics of RL are best understood at the episodic level. Each episode is a cycle of communication between the agent and the environment. The environment represents the problem to be solved, and contains all the information about the problem dynamics. To begin an episode, the agent receives an initial observation from the environment. This observation typically includes variables that define its dynamics, e.g., position and velocity, as well as problem-specific information that can benefit decision making, e.g., relative position to a target.

The initial state received at the beginning of an episode is denoted $\boldsymbol{s_0}$. Based on this state, the agent decides on an action, $\boldsymbol{a_0}$, according to its policy function, $\pi_\theta(\boldsymbol{a_0}|\boldsymbol{s_0})$, where θ represents the weights and biases of the actor neural network. Having communicated this decision to the environment, the agent receives, in response, a reward signal $r_0(\boldsymbol{s_0}, \boldsymbol{a_0})$ and a new state $\boldsymbol{s_1}$. The agent records this state, action, and reward set $(\boldsymbol{s_0}, \boldsymbol{a_0}, r_0)$, and proceeds to choose a new action $\boldsymbol{a_1}$ given $\boldsymbol{s_1}$. This process repeats itself until the environment either indicates that the agent has met a user-defined termination criteria, e.g., colliding with the Moon, or the number of time steps reaches a preset maximum T. At the end of each episode, a trajectory of state-action-reward data is recorded.

During the first episode, the policy is random as the agent explores the available action space. Updating the weights and biases of the agent's neural networks is the only way to improve the underlying policy. After collecting a user-determined amount of data, denoted the 'batch size', the agent performs an optimization update based on its saved state-action-reward trajectory. This update requires calculating an 'advantage', A^π, which is the extent to which the outcome of an action at a specific state is better or worse than expected. To accomplish this computation, the agent must have some internal mechanism by which to estimate the value an observation. This assessment takes the form of a neural network called the 'critic'. Assuming the quality of a state-action pair is measured by its associated reward, the quality of an entire trajectory of state-action pairs, starting from a specific state, is estimated as the sum of discounted future rewards. Hence, the value of a state, $V(\boldsymbol{s_t})$, is computed as,

$$V(\boldsymbol{s_t}) = \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1} r_t\right] \tag{2}$$

where γ is the discount factor that determines the extent to which the agent prioritizes immediate vs. future rewards.

Upon completing an episode, the value of each state in the state-action-reward trajectory is calculated starting with the final state $\boldsymbol{s_T}$ and working backwards. By subtracting the resulting trajectory value, $V(\boldsymbol{s_t})$, by the critic NN's estimated value of each state, an estimate of the advantage is computed. As detailed by Schulman et al., [27] $A^\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ is evaluated as,

$$A^\pi(\boldsymbol{s_t}, \boldsymbol{a_t}) = \left[\sum_{\tau=t}^{T} \gamma^{\tau-t} r(\boldsymbol{s_\tau}, \boldsymbol{a_\tau})\right] - V(\boldsymbol{s_t}) \tag{3}$$

Through the use of the advantage function, the policy of the actor is updated using a version of gradient descent. Similarly, the summation term in Eq. (3) is used to update the value estimate of the critic. Following these updates, a new episode begins with this more accurate pair of NNs. Together, the update strategy is described as a 3 step repeated process in which (1) data is collected, (2) the agent is updated, and (3) new data is collected that solely reflect the performance of the newly updated actor. This strategy, employed in this investigation, is specific to an RL category known as on-policy algorithms.

C. Policy Gradient Methods

Policy gradient methods are a category of RL algorithms that learn a policy function by, using gradient descent, optimizing the parameters that compose the actor NN with respect to the sum of future rewards. Consider a control policy in which the action is sampled from a multivariate distribution $\boldsymbol{a_t} \sim \pi_\theta(\boldsymbol{a_t}|\boldsymbol{s_t})$. This distribution can easily be produced by a NN that outputs the mean and variance of the distribution as a function of the input state. Alternatively, a NN can be used to produce the mean while the variance is directly learned by the agent as a trainable variable. This latter approach is used in this investigation. Fortunately, these two methods are functionally identical in the context of policy gradient methods, since both produce the desired policy distribution $\pi_\theta(\boldsymbol{a_t}|\boldsymbol{s_t})$.

Once defined as a distribution, the policy can be learned by optimizing the actor NN parameters θ with respect to the objective function $J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$. It follows that the update gradient used for the actor is given by $g := \nabla_\theta \left[\sum_{t=0}^{\infty} r_t\right]$. In practice, this gradient is not directly computed, but rather estimated using one of many possible methods detailed by Schulman et al. [27]. One popular form of this estimator is,

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(\boldsymbol{a_t}|\boldsymbol{s_t})\hat{A}_t\right] \tag{4}$$

with the corresponding policy gradient loss function being,

$$L^{PG} = \hat{\mathbb{E}}_t \left[\log \pi_\theta(\boldsymbol{a_t}|\boldsymbol{s_t})\hat{A}_t\right] \tag{5}$$

Optimizing the policy using the loss function given by Eq. (5) is the subject of policy gradient methods

Unfortunately, repeatedly optimizing Eq. (5) using the same state-action-reward trajectory frequently results in destructively large policy updates that prohibit learning an optimal policy [28]. In response to this issue, Schulman et al. designed Trust Region Policy Optimization (TRPO) as an algorithm with guaranteed monotonic improvement [29]. In TRPO, episodes are completed according to a policy $\pi_{\theta_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$. Based on the state-action-reward data that is recorded under this policy, a second policy $\pi_\theta(\boldsymbol{a_t}|\boldsymbol{s_t})$ is optimized, which, in turn, becomes the new $\pi_{\theta_{\text{old}}}$. To achieve the desired monotonic improvement, the Kullback-Leibler (KL) divergence is introduced as an optimization constraint to measure the difference between the two policy distributions, π_θ and $\pi_{\theta_{\text{old}}}$. Together, the optimization scheme under TRPO becomes,

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[R_t(\theta)\hat{A}_t\right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t \left[\text{KL}\left[\pi_{\theta_{\text{old}}}(\cdot|\boldsymbol{s_t}), \pi_\theta(\cdot|\boldsymbol{s_t})\right]\right] \leq \delta \end{aligned} \tag{6}$$

where δ is a limit on the change in policy distributions and $R_t(\theta) = \frac{\pi_\theta(\boldsymbol{a_t}|\boldsymbol{s_t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})}$ is the probability of an action given a certain state under a new, optimized policy divided by the probability of that action under the previous policy. This inequality prevents prohibitively large updates from occurring by ensuring that the change in distribution remains within a specified bound. While TRPO has been successfully applied to many challenging problems, it is a complex algorithm and, thus, prone to implementation errors.

D. Proximal Policy Optimization

Proximal Policy Optimization (PPO) was developed as a simpler and more flexible alternative to achieve the same high performance as TRPO [28]. In its most common formulation, PPO replaces the KL divergence constraint outlined in Eq. (6) with a simple clipping factor contained within its objective function. In this basic formulation, the objective function is given by,

$$L^{CLIP} = \hat{\mathbb{E}}_t \left[\min\left(R_t(\theta)\hat{A}_t, \text{clip}(R_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{7}$$

Here, $R_t(\theta)$ is multiplied by the advantage function, $\hat{A}_t$, which forms the essential component of PPO, $R_t(\theta)\hat{A}_t$. To illustrate this multiplied term, consider the example of an action taken at time step t that resulted in a better than expected outcome. Under such circumstances, the advantage $\hat{A}_t$ will be positive, and the algorithm will seek to make this action more probable under the new policy, i.e., $R_t > 1$. To avoid overly large policy changes, PPO limits the largest possible value of the ratio to $1 + \epsilon$. For example, if $\epsilon = 0.2$, then an action will, at most, be able to become 20% more probable. Similarly, if the action resulted in a poorer than anticipated outcome, $R_t(\theta) < 1$ and a minimum value of $1 - \epsilon$ is enforced.

While clipped PPO is the most common form of the algorithm, an alternative loss function is also available that utilizes the KL divergence in a manner similar to TRPO. This loss is computed as,

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(\boldsymbol{a_t}|\boldsymbol{s_t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})} \hat{A}_t - \beta \text{KL}\left[\pi_{\theta_{\text{old}}}(\cdot|\boldsymbol{s_t}), \pi_\theta(\cdot|\boldsymbol{s_t})\right] \right] \tag{8}$$

where, rather than using a clipping value, a target KL divergence (d_{targ}) is maintained by controlling a penalty coefficient (β). While the precise update rules for updating β are not detailed, the general concept is intuitive. When the KL divergence d is greater than the target value d_{targ}, the penalty coefficient β is increased, which implies that the policy is changing by a greater amount than desired. Hence, increasing β encourages smaller policy updates. Similarly, if $d < d_{\text{targ}}$, β is decreased, and larger policy updates are encouraged. This investigation uses the KL divergence approach to PPO, and the specific implementation employed includes other minor customizations to the algorithm based on Patrick Coady's open source work*.

IV. Dynamical Model

The circular restricted three-body problem (CR3BP) is employed to evaluate the proposed guidance scheme within the context of a complex dynamical regime. In this model, the motion of an object (P_3) is governed by the gravitational influence of two spherically symmetric gravitational bodies (P_1 and P_2), as depicted in Fig. 2. These bodies are assumed to move in circular conic orbits about a common barycenter (B), where 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 $\boldsymbol{r}$, $\boldsymbol{v}$, respectively, comprise the vector $\boldsymbol{\rho}^{\text{spacial}} = [x \;\; y \;\; z \;\; \dot{x} \;\; \dot{y} \;\; \dot{z}]^T$. The vector components are propagated with respect to the system barycenter, B, in a rotating reference frame, denoted by dashed lines in Fig. 2.

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,

$$\ddot{x} - 2\dot{y} - x = -\frac{(1-\mu)(x+\mu)}{r_{13}^3} - \frac{\mu(x-1+\mu)}{r_{23}^3} + a_{\text{lt}} u_x \tag{9}$$

$$\ddot{y} + 2\dot{x} - y = -\frac{(1-\mu)y}{r_{13}^3} - \frac{\mu y}{r_{23}^3} + a_{\text{lt}} u_y \tag{10}$$

$$\ddot{z} = -\frac{(1-\mu)z}{r_{13}^3} - \frac{\mu z}{r_{23}^3} + a_{\text{lt}} u_z \tag{11}$$

$$\dot{m} = 0 + \frac{-f l^*}{I_{\text{sp}} g_0 t^*} \tag{12}$$

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 in a notably sensitive dynamical regime, thus, the proposed guidance strategy exploits the impact of the low-thrust terms to achieve desired behavior. As detailed by Cox et al. [30], thrust direction is defined by the low-thrust acceleration vector,

$$\boldsymbol{a}_{\text{lt}} = \frac{f}{m}\hat{u} = (a_{\text{lt}} u_x)\hat{x} + (a_{\text{lt}} u_y)\hat{y} + (a_{\text{lt}} u_z)\hat{z} \tag{13}$$

where f is the nondimensional thrust magnitude, m is the nondimensional spacecraft mass ratio $M_3/M_{3,0}$, and $\hat{u}$ is the thrust direction unit vector in the rotating frame. The nondimensional thrust magnitude is computed as,

$$f = \frac{F t^{*2}}{l^* M_{3,0}} \tag{14}$$

where F is thrust in kilonewtons and $M_{3,0}$ is the initial mass of the spacecraft. In their formulation, Cox et al. suggest that a nondimensional value of $f \approx 10^{-2}$ in the Earth-Moon system is consistent with existing engine capabilities for Deep Space 1, Dawn, and Hayabusa, while values less than or equal to $f \approx 7\text{e}{-2}$ model a reasonable capability [30].

*Coady, P., "Proximal Policy Optimization with Generalized Advantage Estimation." URL https://github.com/pat-coady/trpo

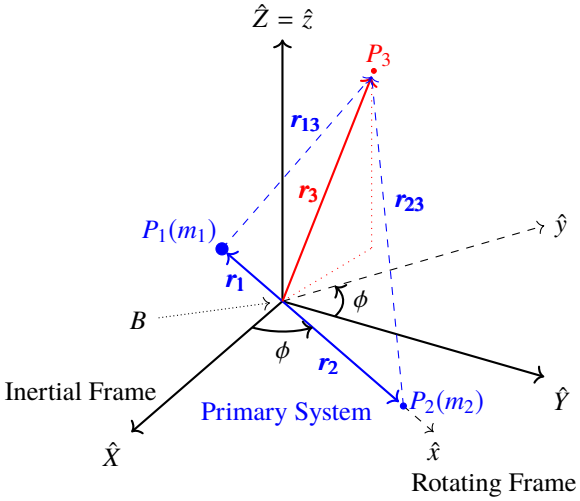


Fig. 2 CR3BP vector definitions where $\mu = 0.2$. The rotating frame $(\hat{x}, \hat{y}, \hat{z})$ is oriented with respect an inertial reference frame $(\hat{X}, \hat{Y}, \hat{Z})$ by an angel $\boldsymbol{\phi}$.

Finally, 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 spacial components $(z, \dot{z}, u_z)$, are all zero. Hence, the vector $\boldsymbol{\rho}^{\text{spacial}}$ becomes $\boldsymbol{\rho} = [x \;\; y \;\; \dot{x} \;\; \dot{y}]^T$, where all the spacial 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,

$$C = 2\left(\frac{1-\mu}{r_{13}} + \frac{\mu}{r_{23}} + \frac{1}{2}\left(x^2 + y^2\right)\right) - \left(\dot{x}^2 + \dot{y}^2 + \dot{z}^2\right) \tag{15}$$

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.

V. Reinforcement Learning Environment

While both the selection and implementation of an appropriate RL algorithm is critical to learning performance, so too is proper design of the RL environment. The environment represents the formulations for the state, action, and reward signals. By definition, 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 to design and is, thus, a critical element of this learning framework. Proper signal design is critical 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.

A. 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 [3]. 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. (9) - (12). Hence, at every time step t, the dynamical state $\boldsymbol{q_t}$ is defined as,

$$\boldsymbol{q_t} = \begin{bmatrix} \boldsymbol{\rho}^{\text{agent}} & m \end{bmatrix} = \begin{bmatrix} x & y & \dot{x} & \dot{y} & m \end{bmatrix} \tag{16}$$

While $\boldsymbol{q_t}$ alone is sufficient to satisfy the Markov property, the agent performance is greatly increased 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. 1. 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 performance and the ability to extrapolate to nearby motion. Including relative state information is similarly useful for RL in a station-keeping environment [31]. The relative information is computed simply as,

$$\boldsymbol{\delta\rho} = \boldsymbol{\rho}^{\text{agent}} - \boldsymbol{\rho}^{\text{ref}} = \begin{bmatrix} \delta x & \delta y & \delta \dot{x} & \delta \dot{y} \end{bmatrix} \tag{17}$$

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 of relative position and velocity. 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,

$$\boldsymbol{\rho}^{\text{ref}} \in \mathcal{R}^{\text{ref}} \qquad \text{s.t.} \qquad k = |\boldsymbol{\delta\rho}| = \sqrt{\delta x^2 + \delta y^2 + \delta \dot{x}^2 + \delta \dot{y}^2} \quad \text{is minimal} \tag{18}$$

This relative state information, along with other optional additional observations, form the complete state signal,

$$\boldsymbol{s_t} = \begin{bmatrix} \boldsymbol{q_t} & \boldsymbol{\delta\rho} & \text{additional observations} \end{bmatrix} \tag{19}$$

of dimension $9 + j$, where j is the number of optional additional observations 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. (15), 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_{ref}, to form the additional observations in Eq. (19). The complete state vector in the CR3BP environment is then defined as $\boldsymbol{s_t} = [\boldsymbol{q_t} \;\; \boldsymbol{\delta\rho} \;\; C_{s_t} \;\; C_{\text{ref}}]$. Omitting the Jacobi constant from the state signal entirely does not prohibit convergence, but the resulting policy is less optimal than one that incorporates the constant. The beneficial impact of including the Jacobi constant indicates that, when applying this approach to other dynamical models where the Jacobi integral may not be available, additional energy-like observations may prove advantageous.

B. Action Definition

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. 1. 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 et al. employed a similar action definition for interplanetary trajectory generation [21]. 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 output 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, actions values are bounded by $[-1, 1]$ and must be 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,

$$f = \frac{\tilde{f}+1}{2} f_{\max} \in [0, f_{\max}] \tag{20}$$

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

$$\boldsymbol{a_t} = \begin{bmatrix} f & u_x & u_y \end{bmatrix} \qquad \text{such that} \qquad \hat{u} = [u_x \;\; u_y] = \frac{[\tilde{u}_x \;\; \tilde{u}_y]}{\sqrt{\tilde{u}_x^2 + \tilde{u}_y^2}} \tag{21}$$

While parameterizing thrust as a unit vector and magnitude is straightforward, a potential drawback is an equation of constraint that is unknown to the controller, i.e., thrust direction is a unit vector. 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 [30] and targeting [32], the bounded action implies that an alternate approach is required for this application.

An alternative low-thrust parameterization that has been applied to PPO by Miller and Linares is empowering the agent to command the thrust in each direction independently, such that each direction is allowed to employ the maximum allowable thrust [20]. 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 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.

C. Reward Signal

The environmental reward is designed to measure 'nearness' to a reference trajectory as a scalar value. This nearness function is modeled as an exponential so that 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, η, 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 deviation exceeds a maximum threshold, a penalty is applied and the episode is terminated. Alternatively, if the agent reaches the target orbit within some position and velocity tolerance, the episode is deemed successful, and an arrival bonus is added to the reward. Together, the reward function is defined as a piecewise function,

$$r = \begin{cases} \eta e^{-\lambda k} & \sqrt{\delta x^2 + \delta y^2} < \delta r_{\max} \;\text{ and }\; \sqrt{\delta \dot{x}^2 + \delta \dot{y}^2} < \delta v_{\max} \\ b & \text{arrival condition met} \\ p_{\mathrm{i}} & \text{impact } P_1 \text{ or } P_2 \\ p_{\mathrm{d}} & \text{deviate from reference} \end{cases} \tag{22}$$

where λ is a scaling factor that increases the gradient of the reward, k is defined in Eq. (18) as the relative distance to the nearest point along the reference, p_{d} is a penalty for deviating from the reference, and p_{i} is a penalty for impacting the primary or secondary body. Finally, η is a scaling term evaluated as,

$$\eta = \frac{i}{n}\xi + 1 \tag{23}$$

where i is the index of the reference trajectory state, n is the size of the set of states along the reference trajectory, $\mathcal{R}^{\mathrm{ref}}$, and ξ is a tuning variable to adjust the rate at which the reward increases along the reference. If the nearest neighbor is along the arrival orbit and not the reference trajectory, then η is assumed to be a maximum value $\eta_{\mathrm{arrival}} = \eta_{\max} = \xi + 1$. Formulating the reward signal to be at maximum when the agent reaches its target encourages the agent to fully complete the given transfer. The reward function employed in this investigation differs from that in Miller and Linares [20] 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.

The measurement of nearness, as detailed in Eq. (22), is visualized in Fig. 3. 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 has occurred, the trajectory is colored light gray to denote areas where a penalty is imposed. Due the exponential term in Eq. (22), 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 η is used in Eq. (22) to encourage the agent to continue along the reference, and discourage the initial periodic behavior. While η discourages initial stationkeeping behavior, it does not entirely eliminate the tendency. Like other optimization methods, abundant local minima encourage an agent to converge to sub-optimal behavior. Including η discourages this behavior, but does nor entirely eliminate the possibility of sub-optimal convergence and return to the stationkeeping local minimum.

D. 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 $O(n)$ to $O(\log n)$, a significant improvement when n is large. In this investigation, Scikit-learn's `sklearn.neighbors.KDTree` implementation is employed to facilitate the nearest neighbor search process [33]. A similar approach is successful for autonomously locating neighbors in higher dimensional Poincaré maps [34, 35]. This application differs in that the neighbor search is conducted for only a single state, rather than intersections from two discrete sets.

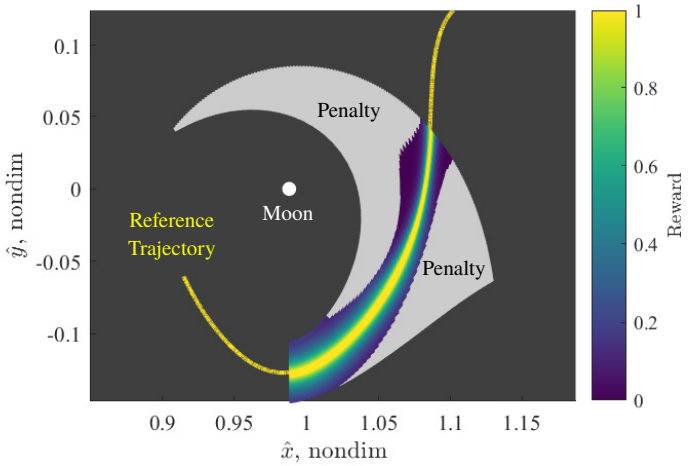


Fig. 3 Motion nearby a reference trajectory originating at the $x = 1 - \mu$ plane. Perturbations are introduced in y_0, and then propagated without thrust. Each state is colored based on the reward function defined by Eq. (22), where $\lambda = 100$, $\eta = 1$, and the maximum deviations in position and velocity are 8000 km and 30 m/s, respectively.

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 complicated 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.

VI. Libration Point Transfer Examples

To illustrate the performance for 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 example cases. The specific characteristic quantities for this system are listed in Table 1. The low-thrust propulsion model assumes a Constant Specific Impulse (CSI) engine with $I_{sp} = 3000$ s, and a propulsion limit $f_{max} = 4 \times 10^{-2}$ nondim. This thrust level corresponds to a spacecraft with more propulsive capability than Hayabusa, Dawn, and Lunar IceCube, but with less than Deep Space 1 [30]. Hence, the sample spacecraft is consistent with current low-thrust capabilities.

For sample scenarios, planar transfers between the L_1 and L_2 libration points, with various geometries, are used as an illustrative test problem for the proposed guidance framework. In particular, Lyapunov orbits at the same Jacobi constant, plotted in Fig. 4(a), are considered. With energy constrained, motion in the lunar vicinity is bounded in configuration space by a forbidden region, frequently denoted the Zero Velocity Curves (ZVCs) [36]. Furthermore, the shared Jacobi constant value between orbits indicates that heteroclinic transfers may be available. Heteroclinic transfers occur when manifold intersections create continuous Δv-free paths between two periodic orbits. Two such transfers appear in Fig. 4(b). 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 [4]. Heteroclinic transfers are one of the few cases in the CR3BP where globally optimal geometries exist, and provide a useful test framework for the controller since no maneuvers are required.

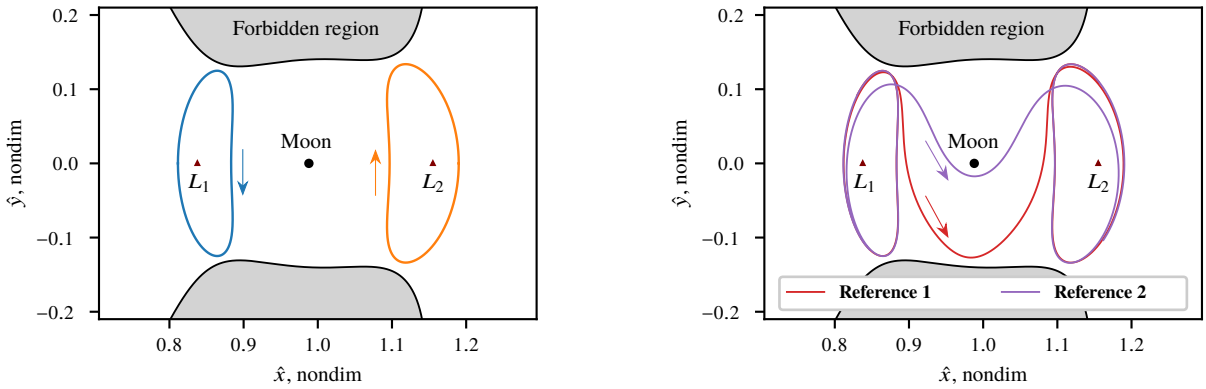
Table 1 Characteristic quantities in the Earth-Moon System

μ, nondim	l^*, km	t^*, s
0.012004715741012	384747.962856037	375727.551633535

While the agent is trained using only one reference trajectory, the controller's ability to generalize control histories to other geometries in the lunar region are investigated. In reality, a variety of factors cause a planned path to change in-flight. For example, on-board targeting yields trajectory corrections and a nearby solution is generated. However, in generating nearby transfers, it is often difficult to produce any initial guess for the control history. In particular, for Orion, Trajectory Correction Maneuvers (TCMs) are nominally zero [7]. However, as perturbations cause a spacecraft to deviate and these maneuvers become necessary, zero thrust is a poor initial guess, and 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 on-board application. To test the controller performance for this generalization, other references and other transfers are examined.

A. Episode Description

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. The nondimensional mass for the initial state is assumed to be one. 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 σ is varied depending on desired disturbance for a particular simulation. Once the initial state is selected, 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



(a) L_1 and L_2 Lyapunov orbits.

(b) Corrected heteroclinic transfers.

Fig. 4 Periodic orbits and heteroclinic transfers used in the sample scenario. All motion at $C = 3.124102$, with the corresponding forbidden regions in gray.

defined in the environment.

Once an initial state with 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 (Δt). The specified Δt between the actions is an important selection for the agent performance. If Δt is too large, then the nonlinearities become more pronounced, and the agent is offered fewer opportunities for sufficient actions over an episode. However, if Δt is too small, there is not time for the thrust direction to demonstrate a discernible impact on the system. For the examples included in this investigation, $t = 0.2$ nondim ≈ 20.87 hrs strikes a balance between the extrema of being too large or too small. After propagating for Δt again, 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.

B. Sample Simulation

To illustrate the resulting PPO-generated controller, a single representative example case is described. An initial state along the L_1 Lyapunov departure orbit is selected, and perturbations of 1106 km and 6.9 m/s are introduced in position and velocity, respectively. In reality, error is computed relative to the reference trajectory rather than the starting orbit, which can add 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. 5, immediately deviates from the reference and impacts the Moon in less than a week.

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 is able to immediately output a control history that returns the spacecraft to its original path and successfully accesses the target L_2 Lyapunov orbit. The new transfer and control history appear in Fig. 6. As expected, the majority of the thrusting for the episode occurs during the initial time intervals as the agent recovers from the introduced perturbation, and subsequent time intervals use 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. 6(a), as well in the periodic sinusoidal thrust magnitude behavior, depicted in Fig. 6(b). For clarity, thrust magnitudes below a user-defined threshold cause the thrust direction magnitude indicator arrows to be omitted from visualization in Fig. 6(a). In this case, 35% is arbitrarily defined. For practical applications, depending on the specific engine characteristics, this threshold possesses physical significance since low-thrust engines only deliver a thrust level within particular bounds. To eliminate the segments where f is small, a differential corrections algorithm could be applied with thrust and coast arcs delineated by the user-defined threshold, as detailed by Das-Stuart et al. [14].

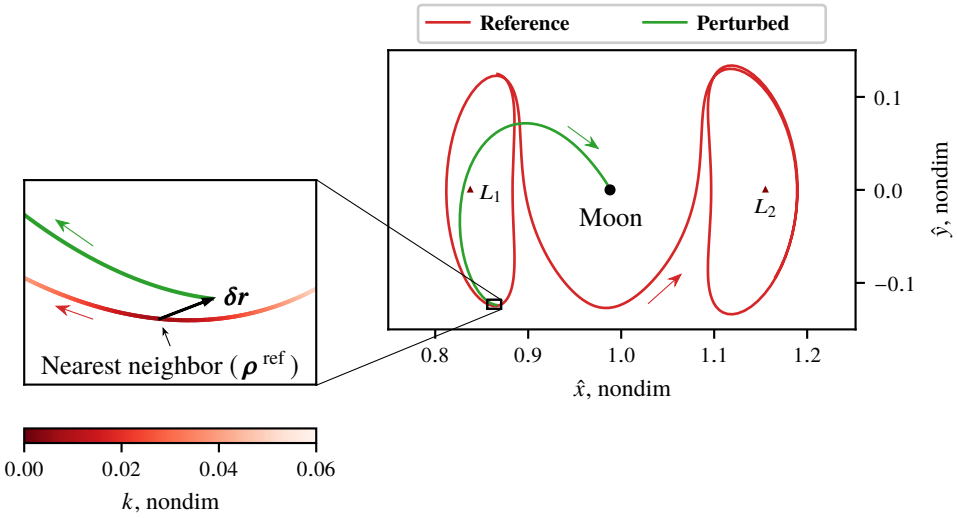


Fig. 5 Sample perturbed initial state that impacts the Moon in 6.4 days. Position perturbation is plotted in configuration space (δr), with magnitude 1106 km. States along the reference trajectory $\mathcal{R}^{\text{ref}}$ appear in the zoomed portion with the shade of red denoting the magnitude of the nearest neighbor distance to the perturbed state, k, defined in Eq. (18)**.**

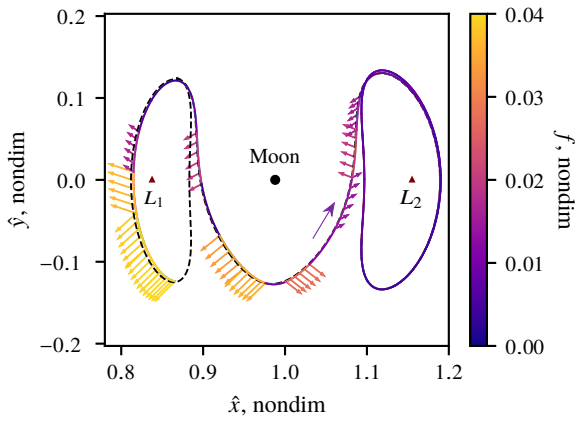
C. Generalization to Other References

If a spacecraft has deviated 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 with a variety of options, including leveraging dynamical systems theory, applying a numerical strategy, and/or employing differential corrections. The methodology for generating new 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. While neural networks in general provide 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 limited by its training experience.

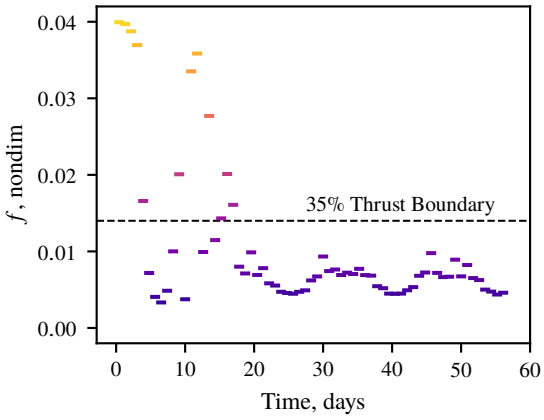
To test the extendability of the PPO-generated controller, several new transfers are examined. First, the case of a new path between the original L_1 and L_2 Lyapunov orbits is demonstrated via the second heteroclinic reference in Fig. 4(b). Next, the reverse of the previous example is also included, in which the agent starts in the L_2 Lyapunov orbit and attempts to track heteroclinic transfers to the L_1 Lyapunov orbit. These two new paths are plotted in Fig. 8(a).

1. New Reference Scenario

To evaluate the case where a new reference is generated in-flight, an alternate transfer between the original orbits is introduced. In particular, without being included in the training process, the controller must now return to Reference 2 in Fig. 4(b). The new path is notably different from the original one; the nearest distance to the Moon along Reference 1 and Reference 2 are 34,546 km and 6,725 km, respectively. Not only is this a large deviation in geometry, significant nonlinearity is added due to the close proximity with the Moon. Hence, this example adds difficulty for the controller, not only in extrapolating to new locations in space, but also demanding that the controller overcome more nonlinearity than was present in training. Returning to the previous example, the perturbation in Fig. 5 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 resulting control history and transfer are plotted in Fig. 7. The agent is clearly able to extrapolate to the new reference by generalizing its experience given the relative state information.

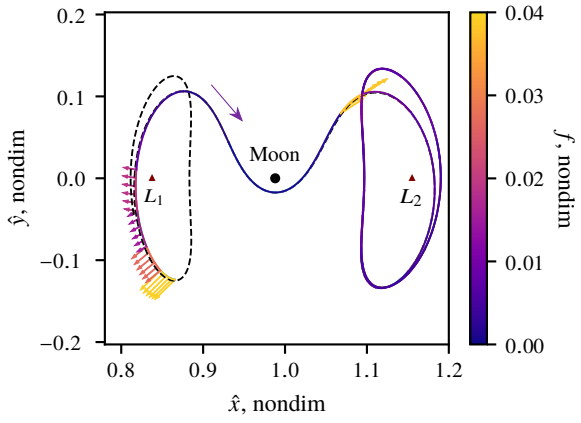


(a) Trajectory and control history.

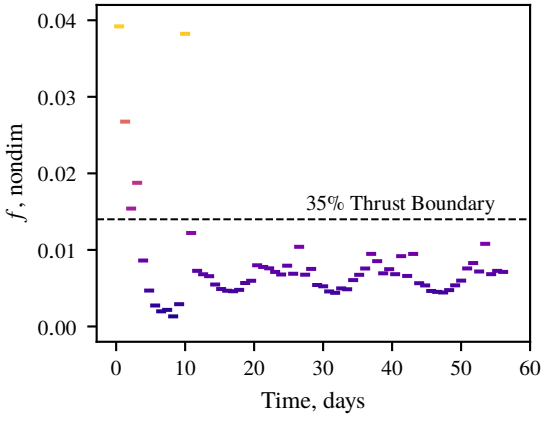


(b) Thrust magnitude history.

Fig. 6 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 (35%), thrust direction indicators are omitted from the control history in (a).



(a) Trajectory and control history.



(b) Thrust magnitude history.

Fig. 7 Resulting control history for previous sample deviation, plotted in Fig. 5, but using Reference 2 from Fig. 4(b). Without additional training, the agent is able to successfully track a new reference trajectory.

2. Reverse Transfer Scenario

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 one of the heteroclinic paths plotted in Fig. 8(a).

A single sample case is again examined. For a perturbation illustrated in Fig. 8(b), the uncontrolled path departs the vicinity of the Moon. Again, 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. 9. These examples demonstrate the RL controller's ability to perform well in challenging scenarios.

D. Monte Carlo Results

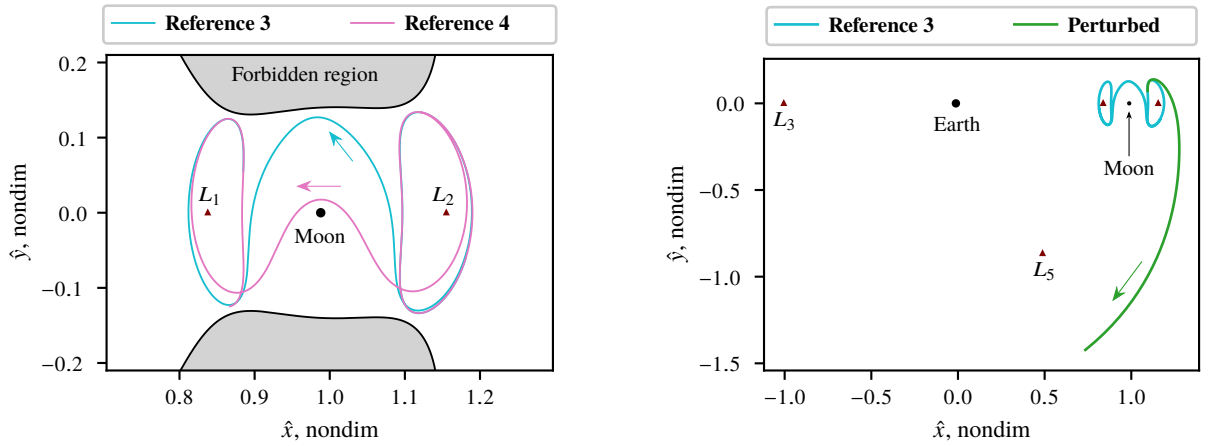
To analyze the performance of a particular controller, many initial conditions with various perturbations are generated and the agent is evaluated based on its output control history. The arrival threshold, characterized in the reward function, offers a means of comparison. 'Arrival' is defined as the relative position and velocity to a state along the arrival orbit with values less than 30 km and 50 cm/s, respectively. For this analysis, three levels of error are investigated. Assuming that orbit determination navigation errors are on the order of 3σ = 1 km and 1 cm/s [31], the proposed controller is tested with errors at 10, 100, and 1000 times the expected navigation error. The 1000× case roughly corresponds to 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.

The result from the Monte Carlo analysis is summarized in Table 2. For Reference 1, each error level yields success in more than 99% of cases. However, as error is increased, trials emerge where the controller is unable to recover. In the 1000× case, 0.77% of perturbations are not successfully recovered. For some of these examples, where the error bounds are 3σ = 1000 km and 10 m/s, it is unreasonable for return to the original motion, and these may correspond to conditions where alternate options should be considered. Four common causes of failure are listed in Table 3. While failure typically indicates deviation, there are also examples where the agent does not meet the arrival criteria within the specified maximum number of time steps. This case does not correspond to deviation and, thus, the arrival percentages listed in 2 are conservative.

To illustrate the failure characteristics that occur for the 1000× case, deviations over time for 100 sample episodes are represented in Fig. 10. The red trajectories correspond to failures where the episode is terminated when either maximum deviation threshold 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 selection of a sub-optimal action which forces the agent to recover from additional error. Failure cases that maintain small amounts of error after 200 days reach arrival criteria if the maximum number of time steps is increased.

The arrival percentages further demonstrate the agent's ability to generalize experience from Reference 1 to the other three geometries. For Reference 2, as the error increases, there is a slight performance degradation compared to Reference 1. However, given the large amount of error, the agent is still able to arrive 98.57% of the time in the 1000× case. This high performance level demonstrates the neural network controller's ability to perform well despite significant uncertainty. For Reference 3, where the spacecraft does not approach the Moon, the agent is able to perform exceedingly well, reaching the arrival criteria more that 99% of the time for all three error levels. Performance decreases slightly when the agent is required to fly nearby the Moon along Reference 4, but is still successful in more than 79% of the scenarios for the three error levels. In consistently satisfying the arrival criteria for this new scenario, the agent demonstrates a remarkable ability to extrapolate 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.

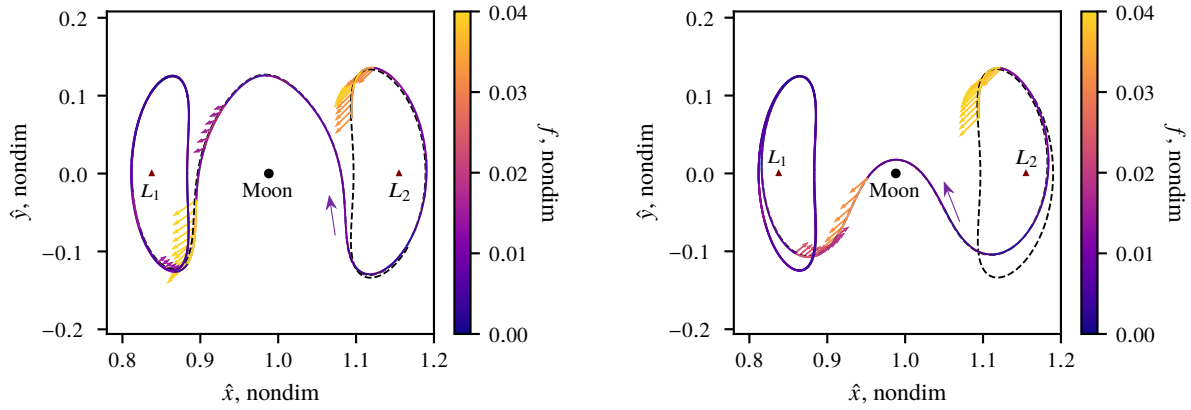
Table 2 also demonstrates the controller's robustness to perturbation levels. Despite the two orders of magnitude difference between error distributions, results remain within 2% for each reference. During training, the agent experiences 1000× error levels and, hence, learns to overcome them. When error is further increased, performance degrades. However, within the demonstrated perturbation magnitudes, there is no consistent correlation between error amount and success percentage across the four references.



(a) Heteroclinic transfers for L_2 to L_1 transfer scenario.

(b) Sample perturbation from L_2 Lyapunov orbit without controller.

Fig. 8 Additional heteroclinic reference trajectories (a) and sample perturbation (b) for evaluating agent's ability to control a reference trajectory in a different direction.



(a) Control history for Reference 3

(b) Control history for Reference 4

Fig. 9 Computed control histories for the L_2 to L_1 reference trajectories plotted in Fig. 8(a), given the perturbation depicted in Fig. 8(b).

Table 2 Monte Carlo results for various reference trajectories with 50,000 episodes each. The agent was only trained using Reference 1, and extrapolates that training to the other reference trajectories.

	L_1 to L_2		L_2 to L_1	
Error Amount	Reference 1	Reference 2	Reference 3	Reference 4
10×	99.79%	99.65%	99.93%	79.35%
100×	99.81%	99.53%	99.92%	79.76%
1000×	99.23%	98.57%	99.78%	79.65%

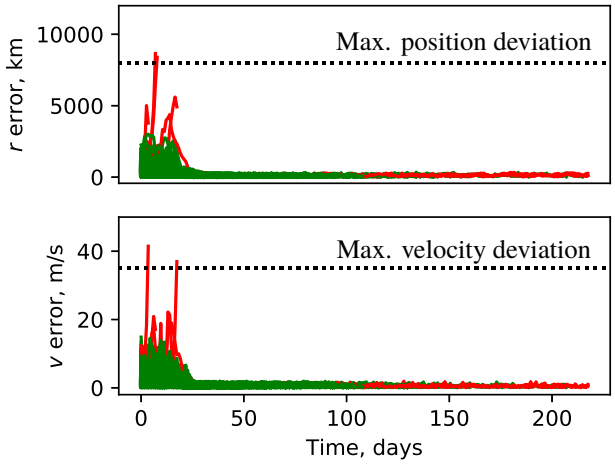


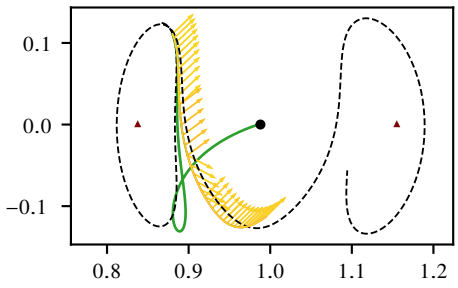
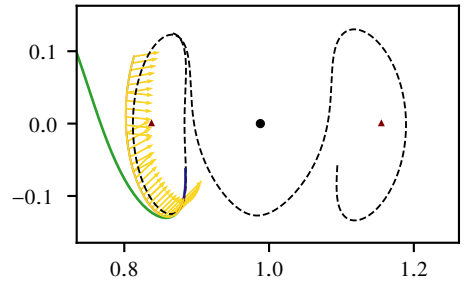
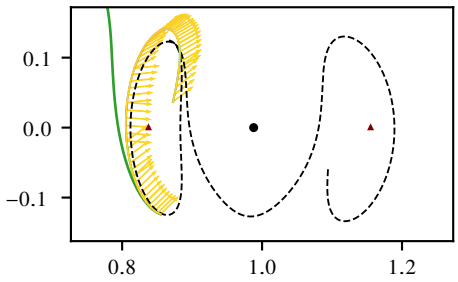
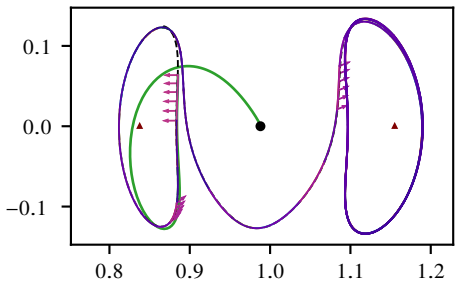
Fig. 10 Deviation over time for 1000 sample episodes using Reference 1. The initial perturbation variance 3σ is 1000 times greater than the expected navigation error. Green denotes episodes that reach the success criteria, whereas red signifies trajectories that cross the maximum deviation threshold in position or velocity, or do not reach arrival conditions within the maximum number of time steps (217.43 days). In the trials, 993/1000 episodes are deemed successful for this sample batch of initial conditions.

E. 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, a large number of simulations are challenging. This investigation focused on this limitation by leveraging super-computing resources, via MIT Supercloud [37], to train many agents simultaneously. 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. In one sample run, 60 identical agents running in parallel produced only two desirable controllers.

For the trained agent used as the controller in the sample scenarios, several sources of failure are present with increased noise. First, for cases where the agent failed to arrive along Reference 1, there are regions of ambiguity that cause a sub-optimal action to produce an unrecoverable trajectory. An example of this behavior is plotted in the first row of Table 3. Here, the agent attempts to prematurely depart the L_1 Lyapunov orbit, which results in an unrecoverable scenario. This ambiguity in thrust direction occurs in regions where two different segments of the reference path are nearby. In these regions, the agent is more likely to implement a poor action since, frequently, only one of the nearby segments is actually accessible. This incorrect thrust direction also demonstrates the importance of the initial action. Due to the introduced perturbation, the first action is critical to performance. An example of a poor initial action yielding an unrecoverable error is depicted in the second row of Table 3. In this case, thrust is needed to recover from the perturbation, but the agent incorrectly implements a nearly-coasting arc that ensures the spacecraft departs the lunar vicinity. The explicit error cause in such examples is challenging to diagnose due to the 'black box' nature of neural networks.

Table 3 Noted sources of failure. In the sample control histories, green denotes the uncontrolled perturbation propagated forward in time.

Failure type	Sample control history ($\hat{x}$-$\hat{y}$, nondim)	Notes
Ambiguous thrust direction		Agent attempts to prematurely depart the L_1 vicinity instead of completing one more revolution around the orbit. For sample case, episode is successful if no initial action is taken.
Poor initial action		After a poor initial action, the deviation becomes unrecoverable with the given maximum thrust level.
Unrecoverable perturbation		Perturbation is such that the agent departs the vicinity of the moon regardless of action choice.
Arrival condition not triggered		Agent completes the transfer, but does not reach the defined arrival criteria of 30 km and 50 cm/s within the maximum number of time steps. For the sample case, in 150 time steps ($\approx$ 130 days), the agent is within 2 km and 2 cm/s of arrival. Given more time steps, arrival criteria is eventually met.

VII. Concluding Remarks

Computationally efficient on-board 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. The demonstrated controller offers computationally efficient on-board guidance in a multi-body regime. By decoupling training from the controller output, this approach utilizes high-performance computers while still producing an algorithm suited to the closed-loop flight environment. The resulting neural network controller is robust to changes in reference geometry, and is able to generalize past experience to new problems. Furthermore, the proposed approach separates the learning agent from the dynamical environment, enabling model-agnostic guidance that is extendable to higher-fidelity domains.

Appendix

A. 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 used in this investigation are summarized in Table 4, with additional suggested values included to indicate cases where desirable behavior was observed given different parameter values. Furthermore, the specific configurations of the employed neural networks are listed in Table 5. These networks were implemented using TensorFlow [38].

Table 4 Suggested parameter values for PPO training and CR3BP RL environment configuration

Variable name	Symbol	Value used	Additional suggested values
Discount factor	γ	0.9	0.85 – 0.9
Number of optimizations on each batch		5	5 – 10
Batch Size		64	32 – 64
Reward steepness	λ	3600	1300 – 4000
Reward arrival bonus	b	25	
Reward scaling gradient	ξ	1	
Reward divergence penalty	p_{d}	-4	
Reward impact penalty	p_{i}	-10	
Actor Learning Rate		0.0001	
Critic Learning Rate		0.0020	
KL Divergence Target	d_{targ}	0.003	
Number of Training Episodes		100,000	>75,000

Table 5 Configuration of actor and critic neural networks employed in this investigation.

		Actor		Critic	
Layer name	Symbol	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`

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 thank you to Rolfe Power for assistance with Python/C++ interfacing. 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.

References

[1] Whitley, R., and Martinez, R., "Options for staging orbits in cislunar space," *2016 IEEE Aerospace Conference*, IEEE, Big Sky, Montana, 2016, pp. 1–9.

[2] Hart, J., King, E., Miotto, P., and Lim, S., "Orion GN&C Architecture for Increased Spacecraft Automation and Autonomy Capabilities," *AIAA Guidance, Navigation and Control Conference and Exhibit*, AIAA, Honolulu, Hawaii, 2008, pp. 1–26.

[3] Sutton, R. S., and Barto, A. G., *Reinfrocement Learning: An Introduction*, 2nd ed., The MIT Press, 2018.

[4] Haapala, A. F., and Howell, K. C., "A Framework for Constructing Transfers Linking Periodic Libration Point Orbits in the Spatial Circular Restricted Three-Body Problem," *International Journal of Bifurcations and Chaos*, Vol. 26, No. 5, 2016.

[5] McGuire, M., Burke, L., McCarty, S., J Hack, K., Whitley, R., C Davis, D., and Ocampo, C., "Low Thrust Cis-Lunar Transfers Using a 40 kW-Class Solar Electric Propulsion Spacecraft," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Stevenson, Washington, 2017, pp. 1–21.

[6] Vavrina, M. A., Englander, J. A., Phillips, S. M., and Hughes, K. M., "Global, Multi-Objective Trajectory Optimization With Parametric Spreading," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Stevenson, Washington, 2017, pp. 1–20.

[7] Marchand, B. G., Weeks, M. W., Smith, C. W., and Scarritt, S., "Onboard Autonomous Targeting for the Trans-Earth Phase of Orion," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 3, 2010, pp. 943–956. https://doi.org/10.2514/1.42384.

[8] Dachwald, B., "Evolutionary Neurocontrol: A Smart Method for Global Optimization of Low-Thrust Trajectories," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Providence, Rhode Island, 2004, pp. 1–16.

[9] De Smet, S., and Scheeres, D. J., "Identifying heteroclinic connections using artificial neural networks," *Acta Astronautica*, Vol. 161, 2019, pp. 192–199.

[10] Parrish, N. L., and Scheeres, D. J., "Optimal Low-Thrust Trajectory Correction with Neural Networks," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Snowbird, Utah, 2018, pp. 1–20.

[11] Furfaro, R., Bloise, I., Orlandelli, M., Di Lizia, P., Tupputo, F., and Linares, R., "Deep Learning for Autonomous Lunar Landing," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Snowbird, Utah, 2018, pp. 1–22.

[12] Grondman, I., Busoniu, L., Lopes, G. A. D., and Babuska, R., "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 42, No. 6, 2012, pp. 1291–1307. https://doi.org/10.1109/TSMCC.2012.2218595.

[13] Das-Stuart, A., Howell, K. C., and Folta, D. C., "A Rapid Trajectory Design Strategy for Complex Environments Leveraging Attainable Regions and Low-Thrust Capabilities," *68th International Astronautical Congress*, Adelaide, Australia, 2017, pp. 1–19.

[14] Das-Stuart, A., Howell, K. C., and Folta, D. C., "Rapid Trajectory Design in Complex Environments Enabled by Reinforcement Learning and Graph Search Strategies," *Acta Astronautica*, Available Online April 25, 2019. https://doi.org/https://dx.doi.org/10.1016/j.actaastro.2019.04.037.

[15] Das-Stuart, A., and Howell, K., "Contingency Planning in Complex Dynamical Environments via Heuristically Accelerated Reinforcement Learning," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Portland, Maine, 2019, pp. 1–21.

[16] Gaudet, B., and Linares, R., "Integrated Guidance and Control for Pinpoint Mars Landing Using Reinforcement Learning," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Snowbird, Utah, 2018, pp. 1–20.

[17] Furfaro, R., and Linares, R., "Waypoint-Based generalized ZEM/ZEV feedback guidance for planetary landing via a reinforcement learning approach," *3rd International Academy of Astronautics Conference on Dynamics and Control of Space Systems, DyCoSS*, Univelt Inc., Moscow, Russia, 2017, pp. 401–416.

[18] Broida, J., and Linares, R., "Spacecraft Rendezvous Guidance in Cluttered Environments via Reinforcement Learning," *29th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Society, Ka'anapali, Hawaii, 2019, pp. 1–15.

[19] Guzzetti, D., "Reinforcement Learning And Topology Of Orbit Manifolds For Station-keeping Of Unstable Symmetric Periodic Orbits," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Portland, Maine, 2019, pp. 1–20.

[20] Miller, D., and Linares, R., "Low-Thrust Optimal Control via Reinforcement Learning," *29th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Society, Ka'anapali, Hawaii, 2019, pp. 1–18.

[21] Miller, D., Englander, J. A., and Linares, R., "Interplanetary Low-Thrust Design Using Proximal Policy Optimization," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Portland, Maine, 2019, pp. 1–16.

[22] Reiter, J. A., Spencer, D. B., and Linares, R., "Spacecraft Maneuver Strategy Optimization for Detection Avoidance Using Reinforcement Learning," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Portland, Maine, 2019, pp. 1–19.

[23] Gaudet, B., Linares, R., and Furfaro, R., "Seeker-based Adaptive Guidance via Reinforcement Meta-learning Applied to Asteroid Close Proximity Operations," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Portland, Maine, 2019, pp. 1–19.

[24] "NASA SBIR 2019 Phase I Solicitation: Deep Neural Net and Neuromorphic Processors for In-Space Autonomy and Cognition," , 2019. URL https://sbir.nasa.gov/printpdf/61636.

[25] Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., and Plank, J. S., "A Survey of Neuromorphic Computing and Neural Networks in Hardware," *CoRR*, Vol. abs/1705.06963, 2017. URL http://arxiv.org/abs/1705.06963.

[26] Bersuker, G., Mason, M., and Jones, K. L., "Neuromorphic Computing: The Potential for High-performance Processing in Space," Tech. rep., The Aerospace Corporation, 2018.

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

[28] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," *CoRR*, Vol. abs/1707.06347, 2017.

[29] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P., "Trust Region Policy Optimization," *CoRR*, Vol. abs/1502.05477, 2015.

[30] Cox, A., Howell, K., and Folta, D., "Dynamical structures in a low-thrust, multi-body model with applications to trajectory design," *Celestial Mechanics and Dynamical Astronomy*, Vol. 131, No. 3, 2019, pp. 1–34.

[31] Guzzetti, D., Zimovan, E. M., Howell, K. C., and Davis, D. C., "Stationkeeping Analysis for Spacecraft in Lunar Near Rectilinear Halo Orbits," *27th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Society, San Antonio, Texas, 2017, pp. 1–24.

[32] York, C. E., and Howell, K. C., "A Two-level Differential Corrections Algorithm for Low-thrust Spacecraft Trajectory Targeting," *29th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Society, Ka'anapali, Hawaii, 2019, pp. 1–20.

[33] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, Vol. 12, 2011, pp. 2825–2830.

[34] Vaquero, M., and Senent, J., "Poincare : A MultiBody, Multi-System Trajectory Design Tool," *7th International Conference on Astrodynamics Tools and Techniques*, Oberpfaffenhofen, Germany, 2018, pp. 1–12.

[35] Pritchett, R., Howell, K. C., and Folta, D. C., "Low-Thrust Trajectory Design for a Cislunar CubeSat Leveraging Structures from the Bicircular Restricted Four-Body Problem," *70th International Astronautical Congress*, Washington D.C., USA, 2019, pp. 1–18.

[36] Bozis, G., "Zero velocity surfaces for the general planar three-body problem," *Astrophysics and Space Science*, Vol. 43, No. 2, 1976, pp. 355–368.

[37] Reuther, A., Kepner, J., Byun, C., Samsi, S., Arcand, W., Bestor, D., Bergeron, B., Gadepally, V., Houle, M., Hubbell, M., Jones, M., Klein, A., Milechin, L., Mullen, J., Prout, A., Rosa, A., Yee, C., and Michaleas, P., "Interactive Supercomputing on 40,000 Cores for Machine Learning and Data Analysis," *2018 IEEE High Performance Extreme Computing Conference (HPEC)*, 2018, pp. 1–6.

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