An Autonomous Stationkeeping Strategy for Multi-Body Orbits Leveraging Reinforcement Learning

Nicholas B. LaFarge[*] and Kathleen C. Howell[†]
Purdue University, West Lafayette, IN, 47907, USA

David C. Folta[‡]
NASA Goddard Space Flight Center, Greenbelt, MD, 20771, USA

Computationally efficient guidance is challenging for stationkeeping applications in nonlinear dynamical regions, where a potential onboard control method must satisfy mission requirements on accuracy, computational footprint, and propellant consumption. Furthermore, determining maneuver locations and frequencies is often accomplished heuristically, with relatively few analytical or numerical tools available to aid in the process. Reinforcement learning is leveraged in this investigation to address both the timing and the control components for low-thrust stationkeeping problems, producing neural network controllers that successfully maintain cislunar periodic orbits without *a-priori* knowledge of effective maneuver placement schemes. Practical examples demonstrate a neural network's ability to 'learn' a suitable maneuver schedule for both machine learning and crossing control schemes alike, resulting in effective maneuver patterns that balance coasting time against propellant consumption, while nevertheless ensuring mission success.

I. Introduction

To establish a sustained human and robotic presence in cislunar and deep space, rapid onboard maneuver planning for orbit maintenance is an essential technology. As applied in this investigation, Reinforcement Learning (RL), a subset of machine learning, is demonstrated as an effective tool for determining both the timing and the control states for low-thrust stationkeeping maneuvers in these challenging regions of space. This approach builds on previous contributions for training a computationally efficient closed-loop controller and demonstrates a novel method for encoding the timing and the location of future maneuvers as learnable neural network parameters, reducing the need for *a-priori* knowledge of maneuver frequency and placement. This approach automatically produces maneuver patterns for any given controller, rendering the learning process for maneuver timing applicable to a broad range of control methods.

Stationkeeping, at a high-level, is comprised of two distinct tasks: control and timing. Depending on mission objectives, previous research efforts suggest a wide variety of orbit maintenance control approaches, including recent investigations that leverage RL to train a neural network controller [1–3]. Once a control approach is selected, an additional challenge is determining suitable locations along the orbit to implement the maneuvers. Too few maneuvers may result in deviation from the reference orbit and/or higher stationkeeping costs, while too many maneuvers adds operational complexity, and may interrupt science objectives. This investigation first demonstrates a simple and flexible training technique for producing a neural network stationkeeping controller and subsequently details a novel approach for 'learning' a suitable maneuver schedule. Separating the planning and control tasks renders the maneuver placement approach applicable to any control method (neural network-based or otherwise) and enables the 'timing' neural network to uncover dynamical regions in the space that lead to low-cost stationkeeping solutions.

For the control task, many current stationkeeping techniques in multi-body dynamical regimes lack global applicability, instead relying on specific characteristics of the underlying problem to construct propellant-efficient maneuvers. For example, Floquet mode control approaches leverage stability information to compute maneuvers that offset the orbit's unstable modes [4]. Alternatively, crossing control methods target certain conditions at crossings of an orbit's plane of symmetry [5]. While powerful and effective, these approaches are often application-specific, and not easily generalized to a wider range of mission scenarios and spacecraft. Alternatively, modern RL techniques offer general approaches to train a neural network controller without direct knowledge of the dynamical environment. Neural

[*]Ph.D. Student, School of Aeronautics and Astronautics, nlafarge@purdue.edu, Student Member AIAA.
[†]Hsu Lo Distinguished Professor of Aeronautics and Astronautics, School of Aeronautics and Astronautics, howell@purdue.edu, AIAA Fellow.
[‡]Senior Fellow, Navigation and Mission Design Branch, david.c.folta@nasa.gov, AIAA Associate Fellow.

networks offer an appealing computationally efficient guidance strategy that is potentially suitable for onboard use. When trained via RL, the separation between the *agent* and the *environment* (i.e., the *controller* and the *controlled system*) provides significant flexibility for spacecraft guidance by removing the reliance on underlying assumptions and characteristics of the dynamical model and mission scenario. Furthermore, the separation between training and closed-loop evaluation produces a computationally efficient controller with attractive onboard characteristics, requiring neither iteration nor numerical integration to operate effectively.

Reinforcement learning has recently emerged as a promising tool in spaceflight guidance applications with potential for onboard use. Previous RL applications are broadly categorized based on the phase of flight. Notably productive areas of RL research are landing problems [6–8] and small body operations [9, 10]. In multi-body problems, RL is applied to transfer guidance along a known reference [11], and to transfer design between multi-body orbits by Das-Stuart et al. [12], Sullivan et al. [13, 14], and Federici et al. [15]. Select additional machine learning-based spaceflight control applications are compiled by Shirobokov et al. [16].

Several previous investigations demonstrate the feasibility of RL as an encouraging novel approach for closed-loop guidance near multi-body libration points. In particular, several authors employ RL to compute stationkeeping maneuvers. Guzzetti first leverages Q-Learning with a discretized state and action space for orbital maintenance of an L_1-Lyapunov orbit in the Earth-Moon system [3]. Alternatively, several authors employ recently developed RL techniques to access and manipulate continuous dynamical environments. Applied to Sun-Earth halo orbits, Molnar applies a Soft Actor Critic (SAC) approach to augment Floquet mode control applied to stationkeeping maneuvers [2], and Bonasera et al. employ Proximal Policy Optimization (PPO) to compute corrective maneuvers that offset momentum unloads in both a Sun-Earth gravitational system and a higher fidelity ephemeris force model [1] . In low-thrust applications, LaFarge et al. apply PPO to produce a controller that maintains both transfers and periodic orbits in the Earth-Moon system [11].

While each previous RL-enabled stationkeeping strategy demonstrates promising results, the training processes often incorporate domain-specific knowledge, limiting their applicability in problems that lack specific characteristics. This investigation builds on prior research by removing domain-specific knowledge from the training process, creating a flexible approach that is more easily applied to new mission scenarios. In particular, previous RL training methods rely on application-specific plane crossing geometries [1, 3] or stability [2] information. The current work introduces a generalization of the training approach to avoid orbit-specific characteristics, producing an RL training algorithm that is applicable to a wider variety of problems, and successfully maintains challenging orbits without relying on relative state error minimization in the training process. For many mission applications, such error minimization is not necessary, and extra propellant is expended to correct what may be an acceptable amount of error. Current multi-body stationkeeping strategies, such as crossing control [5] and Floquet mode [4] methods, achieve low-cost maneuvers by employing alternative approaches that do not rely on minimization strategies as observed in traditional optimal control schemes.

II. Problem Formulation

The proposed RL stationkeeping techniques are evaluated in the Earth-Moon neighborhood. In particular, the Circular Restricted Three-Body Problem (CR3BP) is a useful environment for preliminary evaluation because it both represents a challenging region of space that is relevant to upcoming missions while still admitting truly periodic solutions. Additionally, low-thrust propulsion is included to demonstrate algorithmic performance despite limited control authority and pronounced nonlinearities.

A. Dynamical Model

The CR3BP is a model for the motion of an infinitesimal mass P_3 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 with respect to B, denoted $\boldsymbol{r_3}$, $\boldsymbol{v_3}$, respectively, comprise the vector $\boldsymbol{\rho} = [x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}]^T$. 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 that are powered through solar panels on the spacecraft. Currently, ion thrusters are successfully employed, for example, on Deep Space 1 [17] and

Dawn [18], as well as other missions. Building on this progress, upcoming low-thrust missions include Psyche [19] and the Lunar Gateway [20]. 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 nondimensional CR3BP equations of motion with low-thrust terms,

$$\begin{cases} \ddot{x} - 2\dot{y} - x &= -\dfrac{(1-\mu)(x+\mu)}{r_{13}^3} - \dfrac{\mu(x-1+\mu)}{r_{23}^3} + a^{\text{lt}} u_x \\ \ddot{y} + 2\dot{x} - y &= -\dfrac{(1-\mu)y}{r_{13}^3} - \dfrac{\mu y}{r_{23}^3} + a^{\text{lt}} u_y \\ \ddot{z} &= -\dfrac{(1-\mu)z}{r_{13}^3} - \dfrac{\mu z}{r_{23}^3} + a^{\text{lt}} u_z \\ \dot{m} &= 0 + \dfrac{-f l^*}{I_{\text{sp}} g_0 t^*} \end{cases} \tag{1}$$

where the right column 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 P_3 and 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. [21], thrust direction is defined by the low-thrust acceleration vector,

$$\boldsymbol{a}^{\text{lt}} = \frac{f}{m}\hat{u} = (a^{\text{lt}} u_x)\hat{x} + (a^{\text{lt}} u_y)\hat{y} + (a^{\text{lt}} u_z)\hat{z} \tag{2}$$

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, precession of the rotating frame during thrusting 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, $f = \frac{F t^*}{l^* M_{3,0}}$, where F is thrust in kilonewtons and $M_{3,0}$ is the initial mass of the spacecraft in kilograms. Furthermore, the equivalent ΔV of a low-thrust maneuver is derived from the ideal rocket equation,

$$\Delta V_{\text{equiv.}} = I_{\text{sp}} g_0 \log\left(\frac{m_0}{m_f}\right) \tag{3}$$

and provides an intuitive measure of the change in velocity for a CSI engine.

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 greater 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 [22] (planned for use on Lunar IceCube and LunaH-Map) as well as an 850.2 kg spacecraft with an NSTAR engine [17] (used on Deep Space 1 and Dawn).

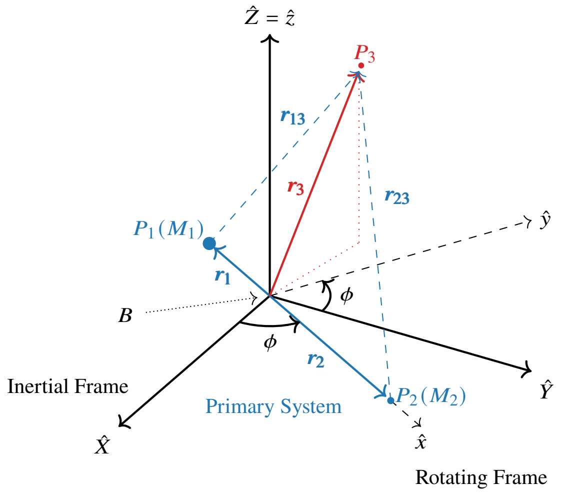


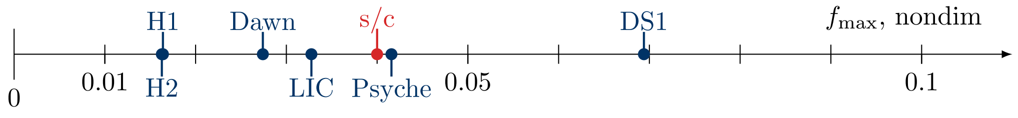
Fig. 1 Vector definitions in the CR3BP. The rotating frame $(\hat{x}, \hat{y})$ is oriented with respect an inertial reference frame $(\hat{X}, \hat{Y})$ by an angle ϕ.

Without a known analytical solution in the CR3BP, numerical integration methods are leveraged to produce a time history stemming from an initial value problem. For implementation, it is useful to express the equations of motion, in

Eq. (1), 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.

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 [23]	$1.640 \cdot 10^{-2}$	510	22.8
H2	Hayabusa 2 [24]	$1.628 \cdot 10^{-2}$	608.6	27.0
LIC	Lunar IceCube [22, 25]	$3.276 \cdot 10^{-2}$	14	1.25
Dawn	Dawn [18]	$2.741 \cdot 10^{-2}$	1217.8	91.0
DS1	Deep Space 1 [17]	$6.940 \cdot 10^{-2}$	486.3	92.0
Psyche	Psyche [19, 26]	$4.158 \cdot 10^{-2}$	2464	279.3
s/c	Sample Spacecraft	$4 \cdot 10^{-2}$	n/a	n/a



B. Cislunar Orbits

Missions to cislunar space are becoming increasingly common. Many upcoming spacecraft plan to leverage periodic orbits in the vicinity of the L_1 and L_2 Earth-Moon CR3BP Lagrange points as baseline trajectories. These missions range from small cubesats, such as Lunar IceCube [25], to the flagship Artemis and Gateway programs [27]. In general, cislunar orbits are computed in the CR3BP during preliminary investigations, and subsequently transferred to a higher-fidelity ephermeris force model. To evaluate preliminary results, the CR3BP serves as the basis for this investigation.

Orbits in the CR3BP are commonly characterized by stability properties. In particular, one commonly used metric computes a linear estimate of stability based on eigenvalues of the monodromy matrix (the state transition matrix propagated for one orbital period). This 'stability index' is evaluated as,

$$\nu = \frac{1}{2}\left(|\lambda_{\max}| + \frac{1}{|\lambda_{\max}|}\right) \tag{4}$$

where $\lambda_{\max}$ represents the maximum eigenvalue of the monodromy matrix. Orbits with $\nu <= 1$ are considered linearly stable. Nearly-stable orbits, such as Near Rectilinear Halo Orbits (NRHOs), are of particular interest to upcoming missions.

Three candidate destination orbits in the lunar vicinity are employed in this investigation to evaluate the proposed stationkeeping framework. The three orbits, visualized in Fig. 2, are summarized as:

1) The 9:2 synodic resonant southern L_2 NRHO (planned baseline for the Lunar Gateway [27]), plotted in red.
2) A distinctly unstable member of the L_2 southern halo orbit family, plotted in blue.
3) An L_2 southern butterfly orbit at nearly 2:1 resonance – similar to the higher-r_p butterfly in Davis et al. [28], plotted in green.

Specific values relevant to each orbit are listed in Table 2. The stability properties for each orbit are particularly important for stationkeeping applications. The stability index, defined in Eq. (4), across the L_2 southern halo and butterfly families are plotted in Figs. 3(a) and 3(b), respectively, where specific colors highlight the stability indices of three sample destination orbits in their respective families.

C. Stationkeeping

Numerous strategies exist for stationkeeping leveraging dynamical structures in the vicinity of libration points. In many cases, specific characteristics of the underlying periodic orbit are exploited to compute low-cost maneuvers, limiting the applicability of any one strategy to a wide theater of operations. For example, *Floquet mode* control

Table 2 Orbit characteristics

	Period, days	C, nondim	Stability ix., ν	Perilune radius, km
9:2 NRHO	6.56	3.046767	1.32	3,210
Halo Orbit	13.59	3.059435	71.02	36,000
Butterfly Orbit	13.64	3.080410	10.03	9,000

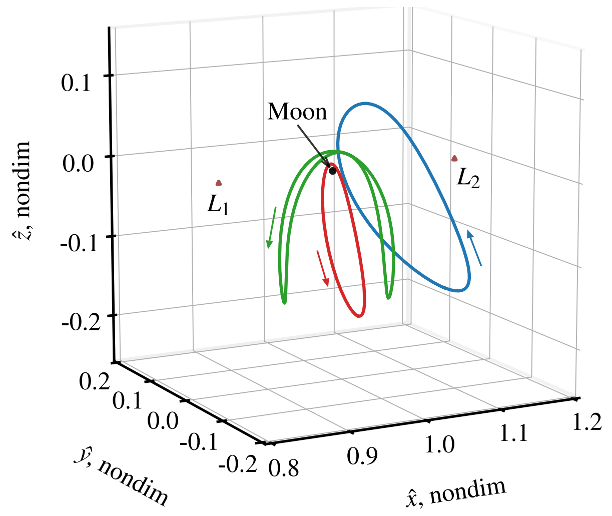


Fig. 2 Sample destination orbits employed in this investigation.

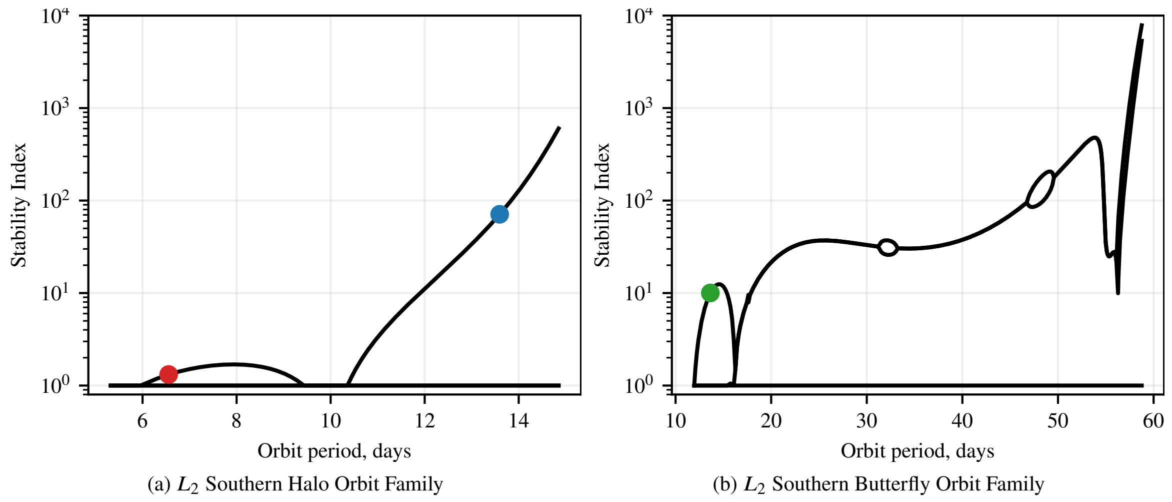


Fig. 3 Stability indices along the Earth-Moon L_2 southern orbit families employed in this investigation, plotted on a logarithmic scale.

exploits stability information for unstable orbits to compute corrective maneuvers that cancel unstable modes [29, 30]. While effective on distinctly unstable orbits, Floquet mode approaches assume accurate computation of the unstable directions: a condition that is frequently unavailable in cislunar space. Numerous alternative strategies are demonstrated for libration point orbits, including classical control theory, Hamiltonian structure preserving control, and continuation strategies [5]. Summaries of previous contributions, available stationkeeping techniques, and mission applications in multi-body regimes are compiled by Shirobokov et al. [31] and Muralidharan [32].

1. Axis Control

One particular stationkeeping approach, crossing control, has emerged as an especially powerful strategy for maintaining symmetric orbits in multi-body regimes [5, 33], with particular recent interest in Earth-Moon NRHO stationkeeping [34]. Many structures in the vicinity of the collinear libration points, including the L_1, L_2, and L_3 halo families, are symmetric across the rotating xz-plane in the CR3BP. The x-axis control algorithm leverages this symmetry by employing a differential corrections technique to target a spacecraft's x-velocity at near-perpendicular crossings of the plane of symmetry at locations downstream. This approach yields remarkably low-cost maneuvers, is effective in a full-ephemeris force model, and is extendable to low-thrust propulsion options [35].

To evaluate the proposed reinforcement learning schemes, a simple x-axis control strategy is implemented. In a full ephemeris-model, x-axis control is comprised of: 1) 'long horizon maneuvers' to compute a long-term virtual baseline solution and 2) 'short horizon maneuvers' to target near-term crossing conditions along the virtual baseline trajectory [5]. This research effort is a preliminary investigation into the proposed RL approaches and the numerical studies are, therefore, limited to the CR3BP. In this context, the periodic orbit itself serves as a long-term baseline, and only short-horizon maneuvers are necessary for the x-axis control algorithm to maintain the orbit. The algorithm itself is detailed by Guzzetti et al. [5], and the resulting impulsive maneuvers are transformed into low-thrust arcs using the ideal rocket equation, as suggested by Newman et al. [35].

2. Maneuver Placement Strategies

Maneuver locations and frequency depend on many factors including, but not limited to, operational constraints, spacecraft hardware, science objectives, navigation and tracking constraints, dynamical sensitivity, impacts to attitude control, and fuel availability [36]. Many applications rely on prior experience, heuristics, and trade studies to determine maneuver placements. Stationkeeping is necessary to maintain periodic orbits about collinear libration points and, therefore, past mission experience provides a useful background for determining maneuver schedules. In Sun-Earth multi-body applications, the ACE mission places maneuvers at one z-axis extremum every 6 months, while WIND and SOHO implement maneuvers at an approximate 90-day cadence [36]. The Genesis mission similarly distributed maneuvers evenly in time, with 2–4 maneuvers implemented per orbital period [37]. In these missions, maximizing the time between maneuvers is particularly important to avoid interruptions to science objectives. In the Earth-Moon system, the ARTEMIS mission implemented maneuvers at xz-plane crossings and y-max amplitudes [38].

In applications involving the upcoming Gateway and Artemis programs, maneuvers along NRHOs are planned near apolune to avoid the dynamically sensitive region near the Moon [35]. For NRHOs, assuming one maneuver per revolution, ΔV placement is determined via Monte Carlo simulations across a range of orbits and maneuver locations [5]. While an exhaustive search of all ΔV locations is effective in this simple case, this approaches relies on discretization and expensive Monte Carlo simulations, becoming quickly intractable for complex mission scenarios and maneuver patterns. Furthermore, the circumstances for different missions and spacecraft influence maneuver planning and, hence, no one pattern of maneuvers is effective for all types of controllers.

An alternative newer methodology for maneuver placement, proposed by Muralidharan and Howell, is to determine stationkeeping locations by observing the angle between maneuvers and downstream stretching directions, computed via singular value decomposition of the state transition matrix [39]. This approach is demonstrated as effective for as many as three maneuver locations per orbit in the L_1 NRHO region, with potential applicability for additional maneuvers.

In reinforcement learning efforts, previous applications place stationkeeping maneuvers either at axis crossings [3], or at regular intervals that are based on previous research efforts [2] or correspond to particular mission characteristics such as the frequency of momentum unload cycles [1]. In other guidance applications, low-thrust spacecraft are assumed to be constantly thrusting, with control parameters adjusted at regular intervals [11, 15]. Implementing maneuvers at a fixed cadence is well-suited to reinforcement learning, but poses several challenges. First, this approach assumes the maneuver frequency is known *a-priori* and, second, the resulting controller may be unable to accommodate shifts in the schedule due to one of many operational constraints.

III. Reinforcement Learning Formulation

A. 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, Twin-Delayed Deep Deterministic Policy Gradient (TD3) [40], is employed in this investigation to train a neural network controller.

1. Neural Networks

A Neural Network (NN) is a class of nonlinear statistical models that are frequently employed in ML classification and regression tasks [41]. 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. Thus, 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 is then processed through an activation function to incorporate nonlinearity into the model [41]. 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, bounded functions are advantageous since they aid in normalizing the output of each neuron. To incorporate nonlinearity, this investigation employs Rectified Linear Unit (ReLU) for all hidden layers, and hyperbolic tangent (tanh) to bound output for the controller. 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.

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 [42]. 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" [43]. However, the proposed neural network controller is relatively small, containing 128,205 trainable parameters and a memory footprint of 513 KB (0.513 MB) assuming 32-bit floating point precision. Furthermore, specialized "neuromorphic" flight processors are currently being developed to enable low-power NN evaluations in space applications [43]. 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 [44].

3. Actor-Critic Reinforcement Learning

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 are accomplished) by seeking to maximize the reward signal. In many cases, terminal conditions exist that cease the learning process. 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. 4. 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* [45].

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 [45]. 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 $< \mathcal{S}, \mathcal{A}, p, r, p_0 >$, where $\mathcal{S}$ and $\mathcal{A}$ are the sets of all possible states and actions, respectively, $p(\boldsymbol{s_{t+1}}|\boldsymbol{s_t}, \boldsymbol{a_t}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow [0, 1]$ is the state-transition probability distribution, $r(\boldsymbol{s_t}, \boldsymbol{a_t}) : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ is the reward function, and $p_0(\boldsymbol{s_0})$ is the density of the initial state distribution.

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 [45], i.e., $p(\boldsymbol{s_{t+1}}|\boldsymbol{s_0}, \boldsymbol{a_0}, \ldots, \boldsymbol{s_t}, \boldsymbol{a_t}) = p(\boldsymbol{s_{t+1}}|\boldsymbol{s_t}, \boldsymbol{a_t})$. 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 labelled 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 $\pi : \mathcal{S} \longrightarrow p(\mathcal{A})$ that is the probability distribution over the action space that maximizes future returns. The expected return is a balance between immediate and future rewards, formalized as,

$$R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(\boldsymbol{s_i}, \boldsymbol{a_i}) \tag{5}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the extent to which the agent prioritizes immediate versus future rewards (typically near 1). Reinforcement learning aims to construct a policy π to maximize the expected return of a task originating from the initial state distribution $p_0(\boldsymbol{s_0})$, thus, maximizing the objective function $J = \mathbb{E}_{\boldsymbol{s_i} \sim p_\pi, \boldsymbol{a_i} \sim \pi} [R_0]$ where p_π is the discounted state visitation distribution following policy π.

The definition of expected return leads to the formalization of the *action-value* function. Value functions are estimated in nearly all RL algorithms and inform the agent of the quality of a particular state. For an MDP, the

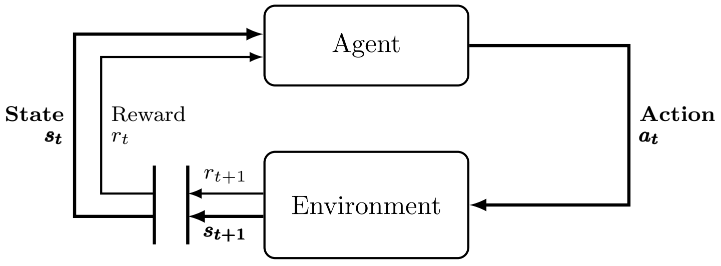


Fig. 4 The agent-environment process in a Markov decision process (reproduced from Sutton and Barto, Ref [45], p.48. Figure 3.1).

action-value function, $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$, is defined as the expected return of taking action $\boldsymbol{a_t}$ at $\boldsymbol{s_t}$, and subsequently following the policy π:

$$Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t}) = \mathbb{E}_{\boldsymbol{s_i} \sim p_\pi, \boldsymbol{a_i} \sim \pi}\left[R_t | \boldsymbol{s_t}, \boldsymbol{a_t}\right] = \mathbb{E}_{\boldsymbol{s_i} \sim p_\pi, \boldsymbol{a_i} \sim \pi}\left[\sum_{i=t}^{T} \gamma^{(i-t)} r(\boldsymbol{s_i}, \boldsymbol{a_i})\right] \tag{6}$$

In practice, many modern algorithms estimate $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ using a 'critic' neural network.

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 some value optimization methods, such as the classical Q-Learning approach, 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) [46], Deep Deterministic Policy Gradient (DDPG) [47], Twin Delayed DDPG (TD3) [40], Soft Actor Critic (SAC) [48], Trust Region Policy Optimization (TRPO) [49], and Proximal Policy Optimization (PPO) [50].

A key delineation between modern actor-critic methods surrounds an assumption concerning the policy distribution from which the actions are sampled. In "on-policy" algorithms, such as TRPO and PPO, update equations assume all actions are sampled from the most recent version of the parameterized policy, $\pi_{\boldsymbol{\theta}}$, causing the current data batch to be discarded following each optimization process. In contrast, 'off-policy" approaches such as TD3 and SAC, actions may be sampled from *any* distribution, allowing for improved learning efficiency. While the state-of-the-art landscape is constantly evolving in reinforcement learning, both on-policy (PPO) and off-policy (TD3 and SAC) schemes are demonstrated as effective in continuous control tasks, and this investigation employs TD3 due to favorable data efficiency properties.

4. Twin Delayed Deep Deterministic Policy Gradient (TD3)

Various RL approaches use the state-action value function $Q(\boldsymbol{s_t}, \boldsymbol{a_t})$, updated by iterating on the Bellman equation, to formulate an effective policy for off-policy control. An early breakthrough for problems with discrete state and action spaces occurred in 1989 with Q-Learning: a temporal difference learning algorithm that allows powerful agents to be trained in an off-policy manor [51]. The core motivation for the Q-Learning approach is simple: if the $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ function is known, that is, the value of all state action pairs is accurately represented, then an optimal policy is available by simply selecting the available action that possesses the largest $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ value. While Q-Learning is very effective, it was limited, along with other RL algorithms at the time, by the assumption of discrete spaces due to the lack of effective nonlinear function approximators.

A major breakthrough in RL came in 2015 when the Q-Learning algorithm was extended to continuous state space problems with the Deep Q-Network (DQN) algorithm [52]. The DQN approach involves the same update rule as Q-Learning, but rather than direct computation, the Q-function is effectively estimated using a neural network: alleviating the need for tabulated results. However, similar to Q-Learning, DQN still suffered from the discrete action space assumption. While acceptable for tasks with a small, finite number of controls (such as Atari games), the curse of dimensionality in the action space limits applicability for continuous control tasks, such as robotics.

To address the action-space dimensionality limitation with DQN, Lillicrap et al. introduce Deep Deterministic Policy Gradient (DDPG) [47]: an extension of DQN and Deterministic Policy Gradient (DPG) [53], that uses a second "actor" neural network to construct a parameterized action function, and perform the $Q(\boldsymbol{s_t}, \boldsymbol{a_t})$ update with the new actor network. A key portion of DDPG is the inclusion of "target" networks for both the actor and value (i.e., *critic*) networks (a similar target approach is also seen in the double-DQN extension of the DQN approach [54]). By including duplicate networks, the second "target" network may be held constant while the actual network is updated. A more stable process results because it doesn't involve continuously updating estimates to be used in the minibatch updates. While powerful, however, DDPG contains several notable limitations. In particular, the value function is often over-estimated, which leads to a significant bias in learning.

Twin Delayed Deep Deterministic Policy Gradient (equivalently, Twin Delayed DDPG, or TD3) [40] is a state-of-the-art off-policy actor-critic algorithm that concurrently learns a parameterized policy $\pi_{\boldsymbol{\theta}}$ and action-value function $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ [40]. Twin Delayed DDPG addresses the instability of DDPG in several ways. First, two separate value networks are included. Incorporating the minimum value estimate of the two networks mitigates the effect of the value function over-estimation bias present in DDPG (along with other actor-critic schemes). Next, DDPG involves *bootstrapping*, i.e., implementing updates based on estimates rather than true values. Noisy estimates cause noisy gradients that pose significant difficulty in network optimization. The TD3 approach seeks to mitigate this error by delaying updates to the actor network in hopes that additional updates to the critic network will provide more accurate estimates for the actor updates. Finally, to avoid peak overfitting in the policy network, policy smoothing is included, which introduces a small amount of clipped random noise to the output of the target policy. Together, these improvements form the TD3 variant of DDPG. While the inclusion of function approximations demands increased complexity, TD3 shares the same core motivation of Q-Learning: optimize a policy network to compute actions that maximize the Q-function. This investigation bases its implementation of TD3 on the open-source "Spinning Up" Tensorflow implementation provided by OpenAI [55], and network architectures are listed in Appendix A, Table 5.

IV. Learning Environment Configurations

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. In general, when constructing RL environments, a balance must be achieved between accurately quantifying the given task, and not including excessive domain-specific knowledge that might render an approach overly application-specific. Furthermore, it is critically 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. 4, 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 for 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.

This investigation involves two distinct learning processes: the 'control' and 'timing' tasks. In the first, given any location along the orbit and various levels of error, the agent is trained to estimate control states that maintain the periodic motion. Next, given a controller (RL-trained or otherwise), the timing environment seeks to train an agent to estimate an effective maneuver placement strategy. While there is some overlap in the characterization of system dynamics across the two environments, each problem supports a different subtask in the stationkeeping problem and, therefore, differs in several important ways, including the transition dynamics and the action definitions.

A. Dynamical Model Characterization

The control and timing tasks are both formulated around stationkeeping and, therefore, share some similarities in this formulation. In particular, both problems necessarily involve identifying a location along a periodic orbit using coordinates that are more easily applied to neural networks. Along with this representation, both control and timing training algorithms share a state signal, quantification of deviation criteria, and an error model.

1. Periodic Orbit Encoding for Neural Networks

Applications in celestial mechanics frequently involve periodic orbits, which presents a specific challenge for neural networks. For two-body Keplerian applications, locations along an orbit are most frequently represented by an angle, e.g., true anomaly, eccentric anomaly, or mean anomaly. In multi-body dynamical regimes, angles are occasionally leveraged in specific applications, but the wide variety in periodic orbit geometry offers challenges in any general angle definition. Instead, elapsed time since a specified initial condition is most frequently employed to represent location along an orbit. Regardless of coordinate choice, directly including cyclic values, such as angle or time along a periodic orbit, in a neural network architecture is ineffective due to the discontinuity at beginning and end conditions. This challenge is further exacerbated by cyclic value extrema typically occurring at apse conditions, that often arise as critically important locations along an orbit. Instead, this investigation leverages a trigonometric ('trig') encoding and decoding [56] process to represent locations along a periodic orbit.

To apply the trig encoding to periodic orbits in the CR3BP, time since a fixed state, $t_{\text{po}} \in [0, \mathbb{P}]$ represents an orbit location, where $\mathbb{P}$ is the period of the underlying structure. The selected fixed initial state may occur anywhere along the orbit. As a cyclic value, t_{po} is conveniently represented as a fraction of the orbit period. The 'encoding' process may be interpreted as projecting t_{po} onto a unit circle, and applying the sine and cosine functions on the resulting angle. 'Decoding' is accomplished via the two-argument arctangent function `atan2`, i.e.,

$$\begin{aligned} &\underline{\textbf{Encoding}}: && t_{\text{po}} \to [\sin\xi, \cos\xi], && \text{where } \xi = \frac{t_{\text{po}} \cdot 2\pi}{\mathbb{P}} \\ &\underline{\textbf{Decoding}}: && [\sin\xi, \cos\xi] \to t_{\text{po}}, && \text{where } t_{\text{po}} = \frac{\texttt{atan2}\,(\sin\xi, \cos\xi) \cdot \mathbb{P}}{2\pi} \end{aligned} \tag{7}$$

The angle ξ has no general physical significance. Several previous applications involving Near Rectilinear Halo Orbits (NRHOs) label ξ, measured from perilune, as 'Mean Anomaly' [39, 57]. This interpretation provides physical insight for orbits with elliptic shapes in the rotating reference frame, but breaks down for more complex geometries that involve multiple periapses in a single period.

The evolution of ξ along each of the sample destination orbits, plotted in Fig. 2, is depicted in Fig. 5. In each case, apolune is arbitrarily selected for $\xi = 0$ and, therefore, $\xi = \pi$ occurs at the half-period of the orbit. Then, $\xi = \pi$ coincides with perilune for the NRHO and halo orbit examples, and with the second apolune for the butterfly orbit. The different evolution of ξ across orbits from different families reinforces ξ's lack of general physical significance in configuration space.

2. 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 [45]. 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 Eq. (1). Hence, at every time step t, the dynamical state $\boldsymbol{q_t} \in \mathbb{R}^7$ is defined as,

$$\boldsymbol{q_t} = \begin{bmatrix} \boldsymbol{\rho}_{\text{s/c}} & m \end{bmatrix} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & m \end{bmatrix} \tag{8}$$

While $\boldsymbol{q_t}$ alone is sufficient to satisfy the Markov property, 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 actor-critic RL formulations, both actor and critic networks receive the complete state signal as inputs, as depicted in Fig. 6. 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 represent valuable data to the agent's performance. The relative information is computed simply as,

$$\boldsymbol{\delta\rho} = \boldsymbol{\rho}_{\text{s/c}} - \boldsymbol{\rho}_{\text{po}} = \begin{bmatrix} \boldsymbol{\delta r} & \boldsymbol{\delta v} \end{bmatrix} = \begin{bmatrix} \delta x & \delta y & \delta z & \delta\dot{x} & \delta\dot{y} & \delta\dot{z} \end{bmatrix} \tag{9}$$

where $\boldsymbol{\rho}_{\text{s/c}}$ is the position and velocity of the agent at some time step, and $\boldsymbol{\rho}_{\text{po}} = \boldsymbol{\rho}(t_{\text{po}})$ is the position and velocity of the nearest neighbor along the given reference path, located at time t_{po} since the specified initial condition. 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 orbit includes a set of n discrete points, $\mathcal{R}^{\text{ref}}$, then the nearest state $\boldsymbol{\rho}_{\text{po}}$ is defined as,

$$\boldsymbol{\rho}_{\text{po}} \in \mathcal{R}^{\text{ref}} \quad \text{s.t.} \quad k = ||\boldsymbol{\delta\rho}|| = \sqrt{\boldsymbol{\delta\rho} \cdot \boldsymbol{\delta\rho}} \quad \text{is minimal} \tag{10}$$

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

$$\boldsymbol{s_t} = \begin{bmatrix} \boldsymbol{q_t} & \boldsymbol{\delta\rho} & \text{additional observations} \end{bmatrix} \in \mathbb{R}^{13+j} \tag{11}$$

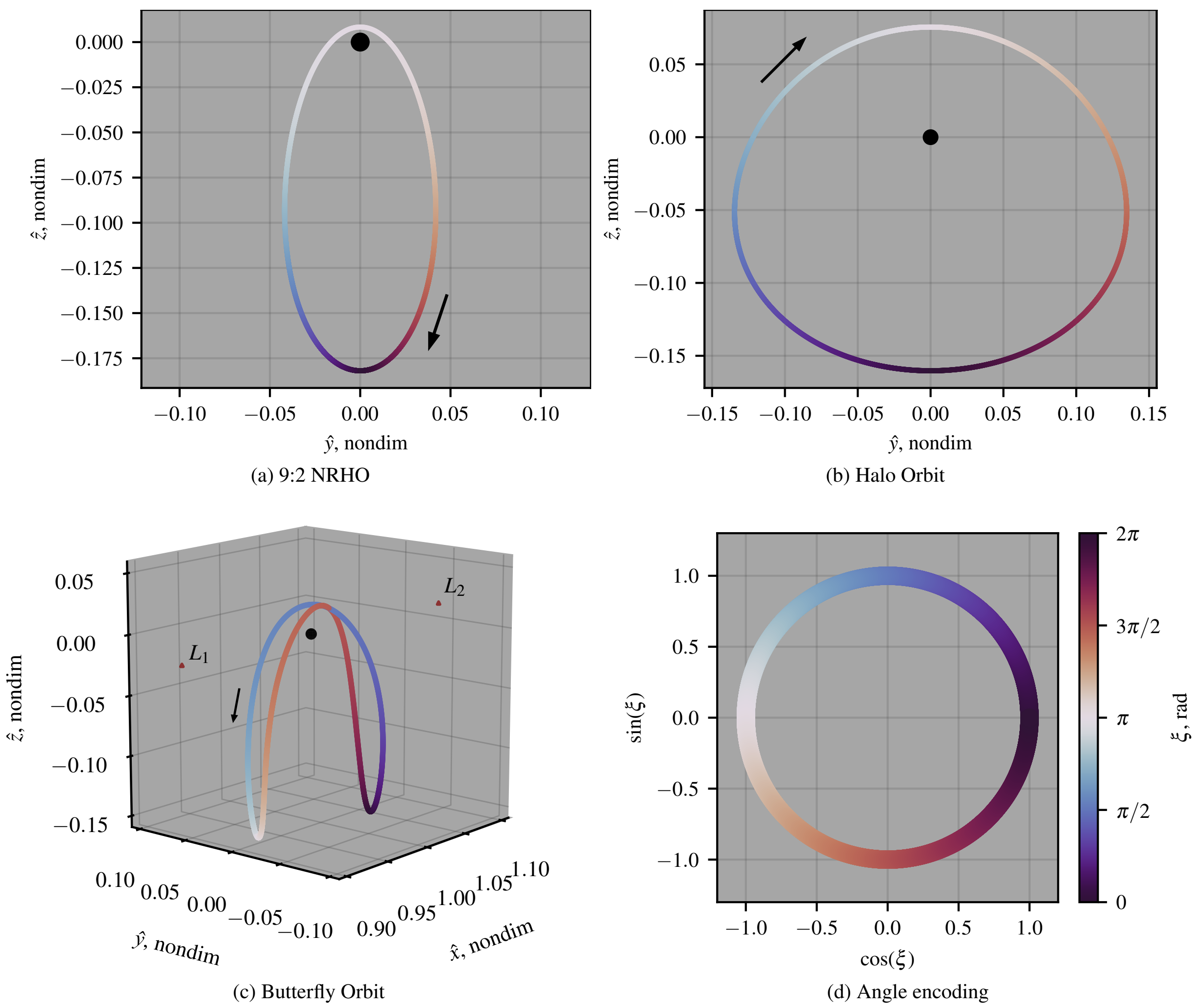


Fig. 5 Locations along each sample orbit encoded by the angle ξ. The visualization of the NRHO (a) and halo orbit (b) are projections onto the rotating yz-plane.

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.

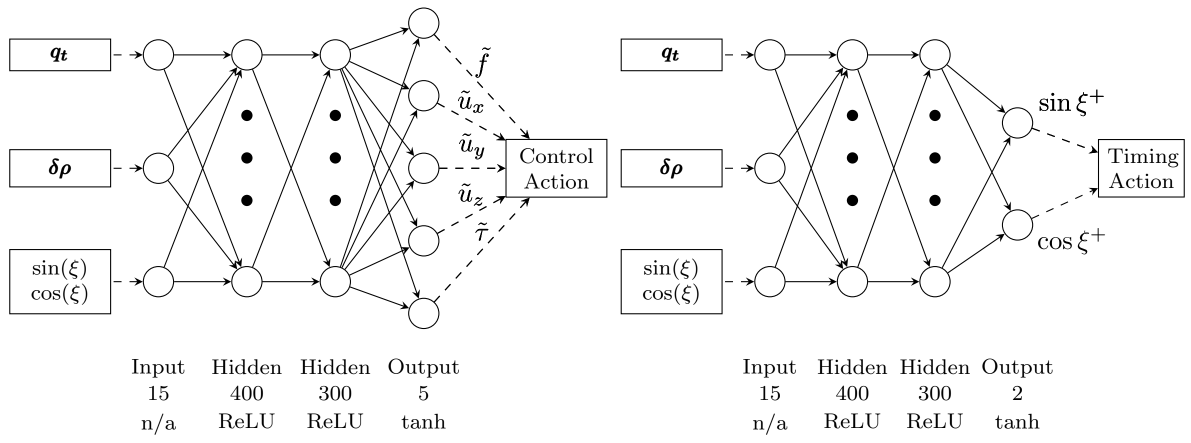
The additional observations are problem-dependent and are, thus, included here as optional parameters. Since this investigation involves periodic reference motion, the location of $\boldsymbol{\rho}_{\text{po}}$ along the underlying periodic orbit is also incorporated into the state signal, located by the angle coordinate ξ as defined in Eq. (7). Leveraging the trig encoding for ξ, the complete state vector becomes,

$$\boldsymbol{s_t} = \begin{bmatrix} \boldsymbol{q_t} & \boldsymbol{\delta\rho} & \sin(\xi) & \cos(\xi) \end{bmatrix} \in \mathbb{R}^{15} \tag{12}$$

Equation (12) forms the state signal for both the control and timing tasks. Depending on the problem, potential additional observations that may improve the learning performance include energy information (quantified by the Jacobi constant) and distance to Moon, r_{23}, however, these possibilities are not investigated here.

3. Deviation Criteria

In formulating both the control and timing training schemes, a measure of deviation is necessary to determine when a spacecraft may be considered "departed" from the underlying periodic orbit. Several factors motivate selecting a



(a) Control (b) Timing

Fig. 6 Actor networks for the (a) control and (b) timing tasks.

criteria. First, training occurs over more than 100,000 episodes, involving as many as 1.5 million interactions with the dynamical environment. Therefore, the deviation flag must be computationally efficient to avoid increasing the already cumbersome computation time. Several implementations are evaluated in this investigation process, including variations of the momentum integral introduced by Guzzetti et al. [5]. While the current results do not yet indicate a benefit to this type of sophisticated early escape detection method, future work may benefit from earlier escape warnings. Instead, this investigation simply computes the absolute deviation in position and velocity from the reference orbit, $|\boldsymbol{\delta r}|$ and $|\boldsymbol{\delta v}|$, respectively, and flags deviations when a user-supplied limit is reached, or the spacecraft impacts a primary body. Deviation is computed from the nearest state along the reference orbit, measured in both position and velocity as a normal, rather than isochronous, correspondence. A K-dimensional tree (KD-Tree) structure is leveraged to efficiently conduct the nearest neighbor search, as suggested by LaFarge et al. [11]. Recall that, in reinforcement learning, the cost function is formulated as the sum of future discounted reward and, therefore, deviation at any point in the future influences prior decisions. Hence, large tolerances are employed, and the algorithm is not sensitive to their precise values. In all applications included in this investigation, the selected tolerances,

$$|\boldsymbol{\delta r}| < 10{,}000 \text{ km} \qquad |\boldsymbol{\delta v}| < 250 \text{ m/s} \tag{13}$$

bound the maximum allowable values for $|\boldsymbol{\delta r}|$ and $|\boldsymbol{\delta v}|$, respectively.

4. Error Models

For preliminary analysis, two sources of error are included in this investigation: orbit insertion and orbit determination. For both the control and timing tasks, episodes begin with a spacecraft inserting into a periodic orbit with some degree of uncertainty. This uncertainty is modeled by constructing $\boldsymbol{\delta\rho}$ measure values from zero-mean normal distributions, where 3σ error bounds are defined as 10 km and 10 cm/s for position and velocity values, respectively. Following insertion, 3σ orbit determination errors of 1 km and 10 cm/s are implemented each time the agent interacts with the environment. While maneuver execution errors are not explicitly implemented during training of the control agent, training actions are computed stochastically and, therefore, resilience to maneuver execution errors occurs naturally via the RL training process.

B. Control Environment

The control environment seeks to train an agent capable of producing stationkeeping maneuvers from anywhere along an orbit in the presence of uncertainty. This process is formalized through three signals that facilitate the agent-environment communications process (state, action, and reward), and the transition dynamics that defines the process by which the environment steps from one state to the next.

- **State** The state signal is summarized in Eq. (12).

- **Action** The action quantifies the three components of a low-thrust maneuver: magnitude $\tilde{f}$, the vector components representing thrust direction, $(\tilde{u}_x,\ \tilde{u}_y,\ \tilde{u}_z)$, and thrust time, $\tilde{\tau}$, as depicted in the output layer of the actor network in Fig. 6(a). 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 control choice given a particular observation, and the variance is included to encourage exploration. 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 bounds 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, the action values are bounded by $[-1, 1]$ and are scaled to reflect actual low-thrust values. Let 'tilde' denote raw, bounded, outputs by the network. As suggested by LaFarge et al. [11], the thrust magnitude is re-scaled by the maximum total allowable nondimensional thrust, and the thrust directions are combined and normalized to form a unit vector,

$$f = \frac{\tilde{f}+1}{2} f_{\max} \in [0, f_{\max}] \qquad \hat{u} = [u_x \ \ u_y \ \ u_z] = \frac{[\tilde{u}_x \ \ \tilde{u}_y \ \ \tilde{u}_z]}{\sqrt{\tilde{u}_x^2 + \tilde{u}_y^2 + \tilde{u}_z^2}} \tag{14}$$

  While zero provides an intuitive lower-bound for thrust time $\tau \in [0, \tau_{\max}]$, the selection of an appropriate upper bound is challenging. If the selected value of $\tau_{\max}$ is too small, the spacecraft may not possess sufficient control authority to maintain the orbit. If the value is too large, small errors in control estimation become exacerbated, and the agent is more likely to converge on a propellant-inefficient policy. A similar challenge arises in computing impulsive maneuvers with bounding the maximum allowable maneuver size. This investigation bases the selection of $\tau_{\max}$ on experiments and the expected stationkeeping maneuver sizes from the literature. Together, the complete action $\boldsymbol{a_t} \in \mathbb{R}^5$ is delivered as,

$$\boldsymbol{a_t} = \begin{bmatrix} f & u_x & u_y & u_z & \tau \end{bmatrix} \in \mathbb{R}^5 \quad \text{where} \quad \tau = \frac{\tilde{\tau}+1}{2}\tau_{\max} \in [0, \tau_{\max}] \tag{15}$$

  An interesting alternative formulation for a planar low-thrust control direction, detailed by Federici et al. [15], leverages a trig encoding similar to Eq. (7), where duplication in the control space is removed by replacing the cosine estimation with a simple sign function, and may be extrapolated to the spatial case by including two additional parameters to model out-of-plane direction. Eliminating duplicate values from the action space may improve learning performance, however this possibility is not investigated.

- **Reward** The reward function forms the feedback signal to communicate the effectiveness of a particular control choice to the agent. Prior investigations involving reference trajectories typically include some form of relative state minimization in the control reward function [1, 2, 5, 11]. While this reward choice encourages the agent to maintain a very close proximity to the reference, such strict adherence to reference motion is often unnecessary in practical applications, and may cause unnecessary propellant spent in correcting permissible levels of error [4]. Instead of minimizing the distance to the underlying orbit, this investigation formulates success as simply "not deviating" from the reference, with a small additional penalty added to encourage propellant-efficient maneuvers (the formalization of 'deviation' is specified in Eq. (13)). Since the expected return is formulated as the sum of future discounted reward, Eq. (5), a deviation penalty, applied at any point, impacts the entire trajectory of preceding actions. This formulation encourages actions that avoid future deviation, thus, encouraging stationkeeping without minimizing state error. The reward is computed as,

$$r = \begin{cases} c - \alpha \Delta V_{\text{equiv.}} & \text{Not deviated from reference} \\ p & \text{Otherwise} \end{cases} \tag{16}$$

  where $c \in \mathbb{R}_{\geq 0}$ is a constant reward for not deviating at each state, $\alpha \in \mathbb{R}_{\geq 0}$ is a scaling factor to determine the extent to which propellant use is penalized, and $\Delta V_{\text{equiv.}}$ is the equivalent ΔV for a low-thrust maneuver, defined in (3). Finally, $p \in \mathbb{R}_{\leq 0}$ is the penalty applied each time the deviation criteria is met. Suggested values are listed in Table 4.

- **Episode process** The agent is trained over either 110,000 episodes or 1.5 million interactions with the environment, whichever occurs first. Each episodes proceeds as follows:
  1) Initial location along the orbit is selected from a uniform distribution where $t_{\text{po}} \sim \mathcal{U}(0, \mathbb{P})$, with an additional perturbation introduced to model an insertion error.

2) A stochastic control action, defined in Eq. (15), is computed by the agent, and simulated in the environment.
3) A ballistic coasting arc is introduced, where propagation time is selected from a uniform distribution where $\Delta t \sim \mathcal{U}(\Delta t_{\text{min}}, \Delta t_{\text{max}})$. Varying propagation time exposes the agent to many location along the orbit, and discourages policies that rely on one particular maneuver frequency. After propagation is complete, an orbit determination error is added to the final state.
4) Steps 2 and 3 alternate until either a the agent reaches a pre-determined maximum number of steps, or the deviation criteria is reached.

Once training is complete, the actor neural network is saved, and may then be leveraged as a deterministic controller. Due to the stochasticity in neural network initialization, it may be necessary that the training process is run several times before a suitable agent is produced.

C. Timing Environment

Given any stationkeeping control strategy, the timing environment trains an agent to estimate the next maneuver location along the reference orbit. The given controller may be an RL-trained neural network or a traditional control strategy. Sample mission applications in this investigation demonstrate timing networks for both RL and x-axis control approaches. The timing agent is rewarded for maximizing the time between maneuvers without escaping from the vicinity of the orbit. The state, action, and reward signals again formulate the learning process:

- **State** The state signal is identical to the control environment, detailed in Eq. (12).
- **Action** The timing agent estimates the location of the next maneuver by leveraging the trig decoding process defined in Eq. (7). Rather than estimating location directly, the neural network is tasked with separately estimating the sine and cosine of the angle ξ^+, which are then decoded to compute the time of the next maneuver, t^+_{po}, measured from a fixed initial state along the periodic orbit, such that

$$\boldsymbol{a_t} = \begin{bmatrix} \sin(\xi^+) & \cos(\xi^+) \end{bmatrix} \in \mathbb{R}^2 \quad \rightarrow \quad t^+_{\text{po}} = \frac{\mathbb{P}}{2\pi} \texttt{atan2}(\sin(\xi^+), \cos(\xi^+))] \tag{17}$$

  The time between successive maneuvers, Δt, is a derived quantity, and is measured as the time between the current (t_{po}) and next (t^+_{po}) maneuver locations, where Δt_{min} serves as a minimum allowable time between thrust segments. If the timing controller suggests a maneuver location such that $\Delta t < \Delta t_{\text{min}}$, then one period of the orbit is added to the coast arc, $\Delta t = \Delta t + \mathbb{P}$. The selection of Δt_{min} is application-dependent, and this investigation employs $\Delta t_{\text{min}} = 24$ hours as the minimum time between maneuvers.
- **Reward** The reward signal for the timing environment is similar to the reward in the control environment, defined in Eq. (16). As modeled previously, a penalty p is imposed for violating the deviation threshold specified in Eq. (13) and a small penalty on propellant expenditure, $\alpha \Delta V_{\text{equiv.}}$, is included to encourage locations that yield low-cost maneuvers. Finally, an additional term $\beta \Delta t$ is added to encourage policies that maximize the time between maneuvers, Δt. The reward is computed as,

$$r = \begin{cases} c - \alpha \Delta V_{\text{equiv.}} + \beta \Delta t & \text{Not deviated from reference} \\ p & \text{Otherwise} \end{cases} \tag{18}$$

  where $\beta \in \mathbb{R}_{\geq 0}$ is a scaling factor that controls the magnitude of the added coasting time bonus. As with the control environment, this formulation of the reward does not explicitly encourage relative state minimization, and instead relies on future deviation penalties to characterize the stationkeeping problem. Suggested values are listed in Table 4.
- **Episode process** The agent is again trained over either 110,000 episodes or 1.5 million interactions with the environment, whichever occurs first. Each episodes proceeds as follows:
  1) Initial location along the orbit is selected from a uniform distribution where $t_{\text{po}} \sim \mathcal{U}(0, \mathbb{P})$, with an additional perturbation introduced to model an insertion error.
  2) The specified controller computes control states given the current navigation state.
  3) The timing agent selects the next maneuver location.
  4) The time between maneuvers is calculated, such that $\Delta t \in [\Delta t_{\text{min}}, \Delta t_{\text{min}} + \mathbb{P}]$.
  5) Propagate Δt and check deviation criteria. If the spacecraft is not deviated, and the agent has not yet reached the pre-determined maximum number of steps, orbit determination errors are added, and steps 2–5 are repeated.

Upon training completion, the actor neural network is saved, and may then be directly leveraged as a deterministic function to compute maneuver locations. Alternatively, experiments demonstrate that agents tend to converge on repeating location sequences that may be represented as simple patterns. These patterns are identified from network simulation, and may be directly employed to avoid neural network evaluation altogether. As with the control environment, stochasticity in neural network initialization often leads to sub-optimal convergence and, hence, the algorithm may need several simulations before a suitable policy is uncovered.

V. Experiments: Cislunar Orbits

The proposed training processes for control and timing tasks are evaluated along the three candidate cislunar destination orbits plotted in Fig. 2. These orbits include the 9:2 synodic resonant L_2 southern NRHO (the planned baseline orbit for the upcoming Lunar Gateway), a more unstable member of the L_2 southern halo family, and an L_2 southern butterfly orbit at nearly 2:1 resonance. Characteristics of each orbit are listed in Table 2, where the stability index is defined in Eq. (4) and visualized in Fig. 3. The variation in orbital period, energy, stability, and geometry between these three orbits provides a useful basis for evaluating the proposed RL approaches.

A. Controller Training

Neural network controllers are trained using TD3 along each of the sample destination orbits. In each case, to overcome the stochasticity in the neural network initialization, and to explore different reward coefficients, many agents are trained in parallel. Once training is complete, a specific controller is selected based on simulation results and desired behavior. Recall that, during training, the next maneuver location is selected from a uniform random distribution between user-specified lower and upper bounds. During training, actions are stochastic, with small perturbations introduced to better explore the action space. After training is complete, performance statistics are gathered on the resulting deterministic controllers. In particular, deviation frequency and propellant consumption are key selection metrics. Then, rather than computing locations randomly, the resulting controller is employed in the timing training process to determine an effective maneuver pattern strategy for the particular controller. For this analysis, a single controller is selected for each mission scenario, with Monte Carlo results listed in Table 3 under the "Training" pattern for each RL controller. The results are summarized as follows,

- **9:2 NRHO**: An inconsistent controller is intentionally selected to evaluate the training process for the timing network. This controller produces relatively propellant-efficient maneuvers, but struggles to maintain the NRHO when maneuvers are located near the Moon. This behavior is consistent with previous research efforts that demonstrate maneuvers near perilune to be ineffective for NRHO stationkeeping [5, 34]. With 1–3 randomly placed maneuvers per revolution, this controller maintains the vicinity of the 9:2 NRHO for 25 revolutions in 79.5% of scenarios.
- **Halo orbit**: A controller is selected that consistently maintains the given halo orbit. In the Monte Carlo simulation with 5–15 randomly-placed maneuvers per revolution, this agent successfully avoids escaping from the reference for 25 revolutions in 100% of trials.
- **Butterfly orbit**: The selected agent consistently maintains the given butterfly orbit, although 1.5% of Monte Carlo trials did escape when maneuvers were randomly placed. This controller produced lower-cost maneuvers than most other trained agents, and is selected for its balance between successfully maintaining the orbit and lower annual stationkeeping cost.

While propellant-efficiency is an important metric in the control task, it is important to note that RL is not an optimal control method and propellant minimization is not expected in the resulting agents. While neural networks are extremely effective at function approximation, the control function is uncovered without any *a-priori* knowledge of desired behavior, and the estimation is never perfectly accurate. Furthermore, controllers in this investigation are expected to compute stationkeeping maneuvers from any location along their respective orbit – a condition that further complicates a particular controller's ability to consistently minimize overall fuel costs. While including domain-specific knowledge in the training process often improves performance in specific applications, it limits applicability to other problems and is, therefore, not included in this investigation. In this application, propellant consumption is considered as one of many metrics when weighing the benefits of neural network control, and is not the only factor in selecting a particular controller from a batch of trained agents.

B. Maneuver Placements and Timing

The timing environment is leveraged to train a controller to select the location for the next maneuver along an orbit, where the maneuver location decoding process is specified in Eq. (17). Again, many agents are trained in parallel, and desirable agents are selected based on several factors, including deviation frequency, number of maneuvers per revolution, and overall propellant consumption. Furthermore, empirical results demonstrate that timing agents tend to converge on specific, repeating, patterns for placing maneuvers. Therefore, the neural network-suggested maneuver sequences may be extracted from simulations and independently analyzed. In this investigation, rather than evaluating the timing network directly in simulations, the computed patterns are employed directly. Initial analysis demonstrates that replacing the neural network with its converged pattern does not degrade overall stationkeeping performance.

To illustrate the pattern identification process, a representative simulation of a converged timing agent along the 9:2 NRHO is plotted in Fig. 7. The episode begins with insertion and, thus, a maneuver is automatically implemented at the initial state. As time progresses in this example, maneuvers fall into a repeating two-maneuver pattern, where colors corresponding to ξ may be correlated to locations along the 9:2 NRHO via Fig. 5(a), and the resulting pattern is visualized in Fig. 8(a). Once identified, the repeating two-maneuver sequence, located by ξ, is directly employed in simulations.

1. 9:2 NRHO Patterns

The timing agent is tasked with determining effective maneuver locations along the 9:2 NRHO leveraging a frequently inaccurate neural network controller. Given random locations, this controller escapes from the reference orbit in more than 20% of simulations. Several timing agents are trained in parallel, and two patterns are extracted from two converged results, plotted in Figs. 8(a) and 8(b). These RL-generated patterns are then compared with an apolune control strategy known to be effective for crossing control techniques [5], depicted in Fig. 8(c). Without *a-priori* knowledge of this behavior, pattern (b) independently uncovers a similar apolune-placement strategy, demonstrating the timing environment's effectiveness in identifying suitable regions of space for maneuver placement.

A Monte Carlo analysis is implemented to evaluate the efficacy of the control strategy given the three specified maneuver patterns, and results are summarized in the "9:2 NRHO" section of Table 3 for Figs. 8(a)–8(c). While the agent frequently diverges given random locations, as implemented in the control training process, the stationkeeping process is significantly stabilized with the identified patterns, successfully maintaining the orbit in more than 97.5% and 95% of simulations for patterns (a) and (b), respectively In both cases, the mean annual stationkeeping cost is approximately 2.3 m/s. This analysis is not intended to represent a practical mission scenario, as a 5% deviation is likely an unacceptable level of error. Instead, this example illustrates the ability of the timing network training process to identify regions of space where a control strategy is particularly ineffective, and place maneuvers elsewhere. Furthermore, the performance difference between the RL-generated one maneuver pattern (b), and the known pattern (c), illustrates the balance imposed during training between deviation and propellant consumption, where the RL-generated pattern leads to lower-cost solutions that deviate more frequently.

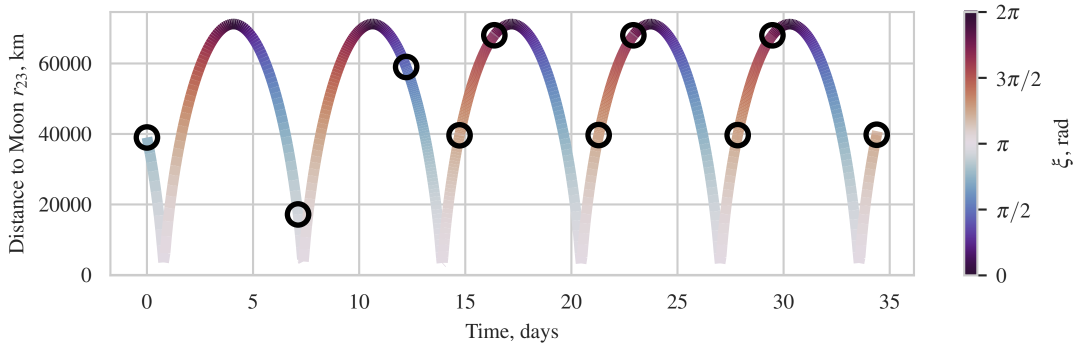


Fig. 7 Distance to the moon over time for a simulation along the 9:2 NRHO, where black circles indicate locations of maneuvers, and color signifies locations along the NRHO, as depicted in Fig. 5(a).

2. *Butterfly Orbit Patterns*

The butterfly orbit, depicted in Fig. 5(c), involves two passes of the Moon during each orbital period and, hence, presents a challenge to the maneuver location selection process. As detailed in Eq. (7), location along the orbit is represented by the angle ξ and, in contrast to halo orbits, the two values $\xi = \{0, \pi\}$ represent separate apolunes. This scenario tests the timing agent's ability to 'learn' an effective pattern despite the more complex evolution of ξ across one orbital period.

A batch of agents is trained along the butterfly orbit, and three patterns are extracted from the resulting converged solutions, depicted in Fig. 9. Patterns (a) and (b) involve four maneuvers per revolution, while (c) implements five. Consistent with the NRHO results, each configuration exhibits a strong preference toward placing maneuvers near apolune. Monte Carlo results leveraging these patterns are listed in the "Butterfly Orbit" section of Table 3. Notably, no escapes occur in the Monte Carlo trials along any of the RL-generated maneuver sequences. Furthermore, each pattern results in a significant annual cost reduction compared to the random maneuver locations employed during training. In particular, pattern (a) possesses a mean annual stationkeeping cost of 18.4 m/s with no failure cases, compared to 35.6 m/s in the training example. Adding the additional maneuver in pattern (c) increases the overall cost, indicating that a four-maneuver cadence is appropriate for this controller-orbit pair.

3. *Halo Orbit Patterns*

The halo orbit employed in this investigation, plotted in Fig. 5(b), is significantly more unstable than the NRHOs and is, therefore, a basis for evaluating the maneuver selection process on a more numerically sensitive dynamical structure. Different agents converge on four representative maneuver patterns, depicted in Fig. 10. Each scenario involves 5 maneuvers per revolution, roughly evenly spaced in time. While a two-maneuver cadence is often employed for halo orbits, the neural network's performance indicates that additional maneuvers are necessary in this application. Mission constraints coupled with the selected control strategy affect the maneuver frequency, as observed in the Genesis mission where up to four maneuvers per orbital period of a Sun-Earth L_1 halo orbit were necessary to avoid deviation cases and satisfy mission requirements for the target point control scheme [37, 58]. As opposed to the NRHO and butterfly orbit scenarios, no specific region of space is outright avoided in the RL-converged patterns, though each pattern demonstrates a slight preference toward placing maneuvers closer to apolune than perilune. Each pattern spaces maneuvers approximately equally in time. This placement strategy is consistent with Genesis, where distributing maneuvers evenly in time was demonstrated as effective for the Sun-Earth L_1 halo orbit [58].

Monte Carlo results for each pattern are specified in the "Halo Orbit: RL control" section of Table 3. Due to overall similarity in the resulting patterns, annual cost remains relatively close between the four cases and, again, in contrast to the NRHO and butterfly orbits, no significant reduction in stationkeeping cost is observed when compared to random maneuver placements (though the total number of maneuvers is reduced from 5–15, to only 5). Of the four included configurations, pattern (a) corresponds to the lowest annual cost of 17.01 m/s. No deviations occur in any of the simulated configurations for the halo orbit scenario.

C. Maneuver Placement: x-axis Control

The timing network training process employs an arbitrary control policy and is, hence, applicable to neural networks and traditional methods alike. To demonstrate this flexibility, an x-axis control scheme for low-thrust is implemented following the process detailed by Newman et al. [35]. To evaluate the RL approach, the timing network is tasked with computing x-axis control maneuver sequences along the L_2 southern halo orbit plotted in Fig. 5(b). Three representative patterns are produced from RL timing agents, and are illustrated in Figs. 11(a)–11(c). The fourth pattern, Fig. 11(d), represents a two-maneuver cadence that alternates between perilune and apolune. This alternating sequence, with additional maneuvers performed at y-max amplitudes, was employed in the ARTEMIS mission [38], and is further demonstrated by Davis et al. in an impulsive x-axis control application applied along a similar halo orbit [34]. The inclusion of a known maneuver schedule provides a useful comparison for the RL-generated pattern results.

Each scenario implements two maneuvers per revolution, and Monte Carlo results are detailed in Table 3 in the section titled "Halo Orbit: x-axis control". Each RL-generated sequence yields lower annual stationkeeping costs, and higher a success percentage, compared to the simple perilune-apolune alternating scheme in Fig. 11(d), with pattern (a) resulting in a mean annual cost of 20 cm/s. Costs are, as expected, notably lower than the RL controller for the same halo orbit, though the computational footprint is two orders of magnitude higher. Despite the large variation between these control schemes, the timing training process is, nevertheless, able to produce effective maneuver sequences for their respective controllers, demonstrating the flexibility of the proposed maneuver sequencing approach, and suggests future applicability to alternative control techniques.

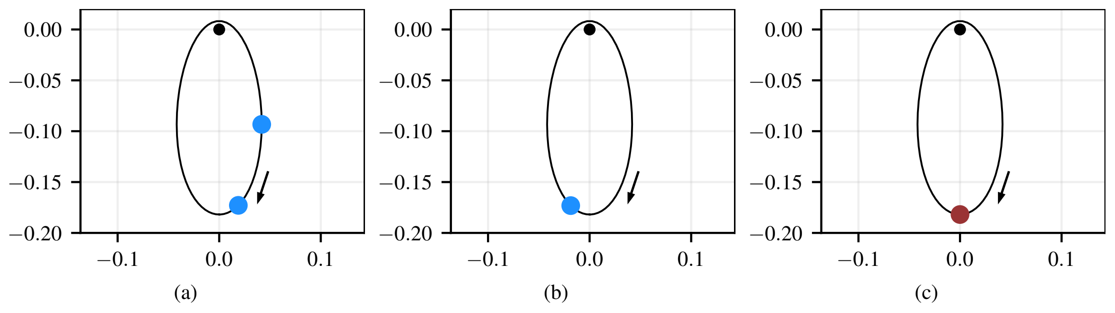


Fig. 8 RL-generated patterns (blue) and the apolune pattern suggested in [5] (red), for a neural network controller along the 9:2 synodic resonant southern NRHO ($\hat{y}$-$\hat{z}$, nondim)

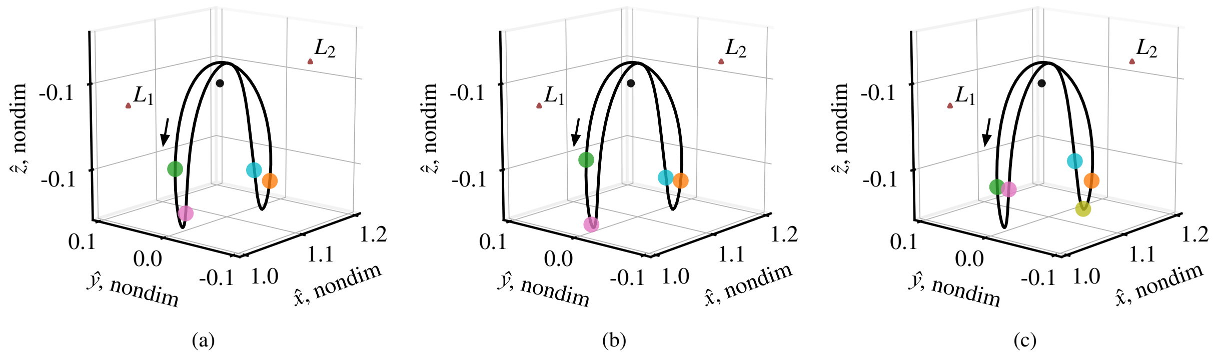


Fig. 9 Neural network control patterns, produced via RL, along the sample butterfly orbit.

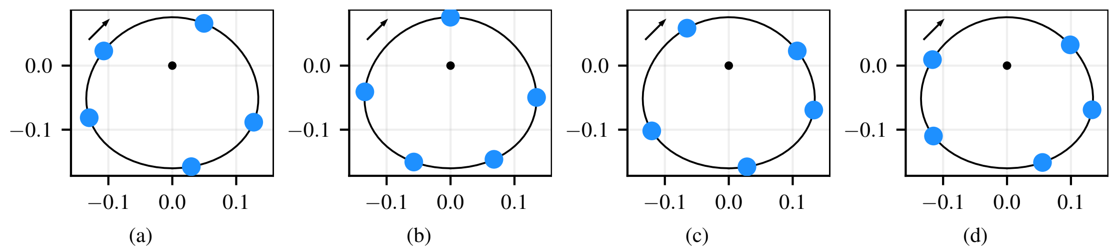


Fig. 10 Sample uncovered patterns for a neural network controller in the sample halo orbit mission scenario ($\hat{y}$-$\hat{z}$, nondim)

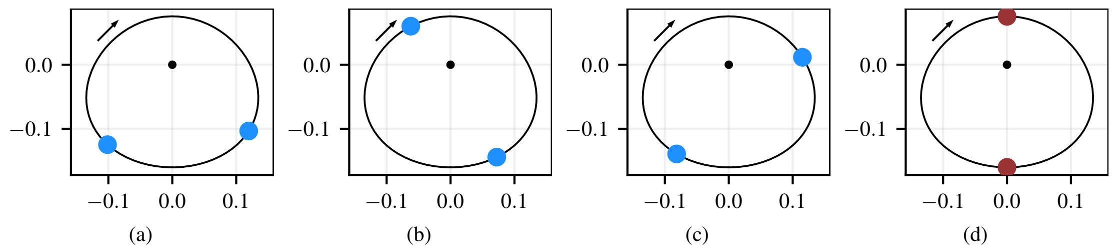


Fig. 11 RL-generated two-maneuver patterns (blue) and an existing configuration from [34] (red) for an *x*-axis control scheme along the sample destination halo orbit ($\hat{y}$-$\hat{z}$, nondim)

D. Multi-Period Patterns

While the majority of the timing agents converge on patterns that repeat every revolution, several cases arise where an agent produces maneuver location sequences that repeat at period multiples. Two such patterns are depicted in Fig. 12 and Fig. 13, where the former repeats every two periods along the NRHO leveraging an RL controller, and the latter repeats every three periods of the halo orbit employing an x-axis control scheme. Monte Carlo simulation results for each pattern are listed in Table 3, demonstrating that both cases are equal to, or better than, other uncovered maneuver sequences in both success percentage and annual cost (though neither performs dramatically better). These non-homogeneous patterns demonstrate the neural networks' ability to uncover and estimate more complex maneuver arrangements, and suggest this methodology is applicable in more challenging domains where such complexity is beneficial for the given problem.

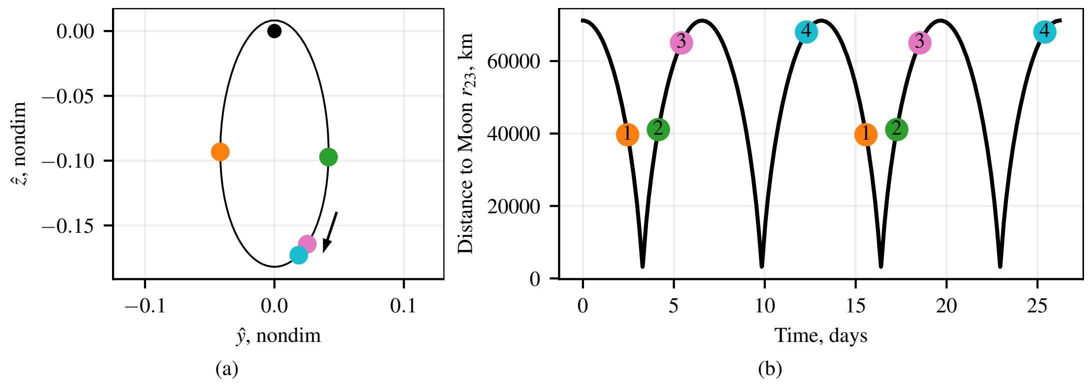


Fig. 12 Sample non-homogeneous RL-generated maneuver pattern for an RL controller along the 9:2 NRHO

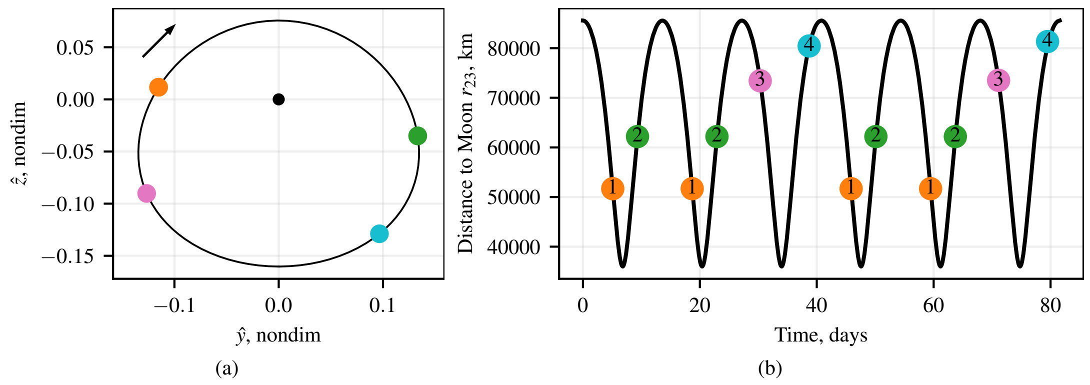


Fig. 13 Maneuver sequence that repeats for every three periods of the sample halo orbit, employing an x-axis control scheme

Table 3 Monte Carlo results with 200 simulations and 25 orbit periods for each cislunar orbit plotted in Fig. 2, where "success" indicates the controller did not deviate from the reference orbit. In cases where RL-control is employed, "Training" indicates the random maneuver location selection process that is employed during training of the RL controller.

	Pattern Fig. No.	**ΔV/ Rev.**	**Success Pct.**	**Annual Cost**		**Pattern**
				μ, m/s	σ, m/s	ξ, rad
9:2 NRHO RL control	Training	1–3	79.5%	5.66	1.37	Random
	Fig. 8(a)	2	97.5%	2.32	1.78	[3.9291, 5.4982]
	Fig. 8(b)	1	95%	2.35	0.76	[0.7741]
	Fig. 8(c)	1	98%	3.16	0.83	[0]
	Fig. 12	3,1	98%	2.32	0.76	[2.3545, 3.9756, 5.1958, 5.4999]
Butterfly Orbit RL control	Training	5–15	98.5%	35.60	5.05	Random
	Fig. 9(a)	4	100%	18.40	0.62	[0.7833, 2.2087, 3.6375, 5.4107]
	Fig. 9(b)	4	100%	21.76	3.79	[0.7858, 2.1442, 2.9305, 5.4986]
	Fig. 9(c)	5	100%	28.33	1.78	[0.7854, 2.3570, 3.9278, 5.3160, 6.2392]
Halo Orbit RL control	Training	5–15	100%	20.18	1.80	Random
	Fig. 10(a)	5	100%	17.01	0.91	[1.5419, 2.4496, 3.4051, 4.8138, 6.0126]
	Fig. 10(b)	5	100%	22.41	0.97	[0.5345, 1.9185, 3.1408, 4.4414, 5.6436]
	Fig. 10(c)	5	100%	19.33	1.39	[1.3195, 2.7830, 3.8332, 4.6227, 6.0247]
	Fig. 10(d)	5	100%	23.71	0.99	[1.2235, 2.3373, 3.7519, 4.6194, 5.7684]
Halo Orbit x-axis control	Fig. 11(a)	2	100%	0.20	0.030	[1.0233, 4.9817]
	Fig. 11(b)	2	100%	0.23	0.036	[2.8031, 5.5979]
	Fig. 11(c)	2	99.5%	0.21	0.042	[0.7854, 3.9270]
	Fig. 11(d)	2	99.0%	0.24	0.040	[0, π]
	Fig. 13	2,2,2	100%	0.20	0.025	[2.3562, 4.3136, 2.3562, 4.3136, 1.4497, 5.3210]

VI. Concluding Remarks

Reinforcement learning offers a powerful approach for uncovering insight in support of stationkeeping for timing and control along challenging, nonlinear multi-body orbits. This investigation offers a general approach for training a neural network stationkeeping controller that does not rely on individual orbit characteristics or domain-specific knowledge. Furthermore, reinforcement learning is demonstrated as remarkably effective in determining maneuver placement strategies for neural network and x-axis control methods alike, suggesting future applicability for complex destination orbits and alternative control schemes. While computational efficiency and accuracy motivate neural network control applications, further research is necessary to address current propellant consumption limitations. Onboard stationkeeping planning will enable future spacecraft to operate autonomously, and neural networks provide an exciting component in facilitating autonomy despite challenging nonlinear regions of space.

Appendix

A. Parameter Selection

Parameter selection and tuning is an important aspect in both TD3 learning and RL environment design. Well-performing agents may be produced with different combinations of parameters. Suggested values employed in this research are summarized in Table 4. Furthermore, the specific configurations of the employed neural networks are listed in Table 5. These networks are implemented using TensorFlow [59].

Table 4 Suggested parameter values for RL and environment configurations

Variable name	Symbol	Equations	Suggested values
Discount factor	γ	Eqs. (5) and (6)	0.99
Actor Learning Rate			0.0001
Critic Learning Rate			0.001
Reward constant	c	Eqs. (16) and (18)	0–1
Reward divergence penalty	p	Eqs. (16) and (18)	-10
Reward control penalty coefficient	α	Eqs. (16) and (18)	0–1
Timing reward coasting coefficient	β	Eq. (18)	0.1–5 ($> \alpha$)
Maximum thrusting time	$\tau_{\max}$	Eq. (15)	1–3 hrs

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

	Control Actor		Control Critic		Timing Actor		Timing Critic	
Layer name	Size	Activation	Size	Activation	Size	Activation	Size	Activation
Input layer	15	-	15	-	15	-	15	-
Hidden 1	400	ReLU	400	ReLU	400	ReLU	400	ReLU
Hidden 2	300	ReLU	300	ReLU	300	ReLU	300	ReLU
Output	5	tanh	1	linear	2	tanh	1	linear

Acknowledgments

The authors thank the Purdue University School of Aeronautics and Astronautics for facilities and support in conducting this research. Special thanks to Kenza Boudad for helpful feedback on visualization techniques. This work was supported by a NASA Space Technology Research Fellowship, NASA Grant 80NSSC19K1175.

References

[1] Bonasera, S., Elliott, I., Sullivan, C. J., Bosanac, N., Ahmed, N., and McMahon, J., "Designing Impulsive Station-Keeping Maneuvers Near a Sun-Earth L2 Halo Orbit via Reinforcement Learning," *31st AAS/AIAA Spaceflight Mechanics Meeting*,

AAS/AIAA, Charlotte, North Carolina (Virtual), 2021.

[2] Molnar, A. B., “Hybrid Station-Keeping Controller Design Leveraging Floquet Mode and Reinforcement Learning Approaches,” Master’s thesis, Purdue University, Dec. 2020.

[3] 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.

[4] Folta, D. C., Pavlak, T. A., Haapala, A. F., Howell, K. C., and Woodard, M. A., “Earth–Moon libration point orbit stationkeeping: Theory, modeling, and operations,” *Acta Astronautica*, Vol. 94, No. 1, 2014, pp. 421–433. https://doi.org/https://doi.org/10.1016/j.actaastro.2013.01.022.

[5] 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.

[6] Furfaro, R., Scorsoglio, A., Linares, R., and Massari, M., “Adaptive generalized ZEM-ZEV feedback guidance for planetary landing via a deep reinforcement learning approach,” *Acta Astronautica*, Vol. 171, 2020, pp. 156–171.

[7] Gaudet, B., Linares, R., and Furfaro, R., “Deep reinforcement learning for six degree-of-freedom planetary landing,” *Advances in Space Research*, Vol. 65, No. 7, 2020, pp. 1723–1741.

[8] Cheng, L., Wang, Z., and Jiang, F., “Real-time control for fuel-optimal Moon landing based on an interactive deep reinforcement learning algorithm,” *Astrodynamics*, Vol. 3, No. 4, 2019, pp. 375–386.

[9] Gaudet, B., Linares, R., and Furfaro, R., “Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations,” *Acta Astronautica*, Vol. 171, 2020, pp. 1–13.

[10] Gaudet, B., Linares, R., and Furfaro, R., “Six Degree-of-Freedom Hovering over an Asteroid with Unknown Environmental Dynamics via Reinforcement Learning,” *20th AIAA Scitech Forum*, AIAA, Orlando, Florida, 2020. https://doi.org/10.2514/6.2020-1910, URL https://arc.aiaa.org/doi/abs/10.2514/6.2020-1910.

[11] LaFarge, N. B., Miller, D., Howell, K. C., and Linares, R., “Autonomous Closed-Loop Guidance using Reinforcement Learning in a Low-Thrust, Multi-Body Dynamical Environment,” *Acta Astronautica*, Vol. 186, 2021, pp. 1–23. https://doi.org/https://doi.org/10.1016/j.actaastro.2021.05.014.

[12] 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*, Vol. 171, 2020, pp. 172–195.

[13] Sullivan, C. J., Bosanac, N., Anderson, R. L., Mashiku, A. K., and Stuart, J. R., “Exploring Transfers between Earth-Moon Halo Orbits via Multi-Objective Reinforcement Learning,” *2021 IEEE Aerospace Conference*, IEEE, 2021, pp. 1–13. https://doi.org/10.1109/AERO50100.2021.9438267.

[14] Sullivan, C. J., Bosanac, N., Mashiku, A. K., and Anderson, R. L., “Multi-Objective Reinforcement Learning for Low-Thrust Transfer Design between Libration Point Orbits,” *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Big Sky, Montana (Virtual), 2021.

[15] Federici, L., Scorsoglio, A., Zavoli, A., and Furfaro, R., “Autonomous Guidance for Cislunar Orbit Transfers via Reinforcement Learning,” *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Big Sky, Montana (Virtual), 2021.

[16] Shirobokov, M., Trofimov, S., and Ovchinnikov, M., “Survey of Machine Learning Techniques in Spacecraft Control Design,” *Acta Astronautica*, Vol. 186, 2021, pp. 87–97.

[17] Rayman, M. D., Varghese, P., Lehman, D. H., and Livesay, L. L., “Results from the Deep Space 1 technology validation mission,” *Acta Astronautica*, Vol. 47, No. 2, 2000, pp. 475–487.

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

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

[20] Irimies, D., Manzella, D., and Ferlin, T., “Summary of Gateway Power and Propulsion Element (PPE) Studies,” *2019 IEEE Aerospace Conference*, IEEE, Big Sky, Montana, 2019, pp. 1–6.

[21] 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.

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

[23] Kuninaka, H., Nishiyama, K., Funaki, Y. S. I., and Koizumi, H., “Hayabusa Asteroid Explorer Powered by Ion Engines on the way to Earth,” *31st International Electric Propulsion Conference*, Ann Arbor, Michigan, 2009, pp. 1–6.

[24] Nishiyama, K., Hosoda, S., Ueno, K., Tsukizaki, R., and Kuninaka, H., “Development and Testing of the Hayabusa2 Ion Engine System,” *Transactions of the Japan Society for Aeronautical and Space Sciences*, Vol. 14, No. 30, 2016, pp. 131–140.

[25] Bosanac, N., Cox, A. D., Howell, K. C., and Folta, D. C., “Trajectory design for a cislunar CubeSat leveraging dynamical systems techniques: The Lunar IceCube mission,” *Acta Astronautica*, Vol. 144, 2018, pp. 283–296.

[26] Snyder, J. S., Lenguito, G., Frieman, J., Haag, T., and Mackey, J., “The Effects of Background Pressure on SPT-140 Hall Thruster Performance,” *53rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, AIAA, Cincinnati, Ohio, 2018.

[27] Lee, D. E., “Gateway Destination Orbit Model: A Continuous 15 Year NRHO Reference Trajectory,” White paper, NASA Johnson Space Center, Aug. 2019.

[28] Davis, D. C., Phillips, S. M., Howell, K. C., Vutukuri, S., and McCarthy, B. P., “Stationkeeping and Transfer Trajectory Design

for Spacecraft in Cislunar Space," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Columbia River Gorge, Stevenson, Washington, 2017, pp. 1–20.

[29] Simó, C., Gómez, G., Llibre, J., and Martínez, R., "Station keeping of a quasi-periodic halo orbit using invariant manifolds," *Second International Symposium on Spacecraft Flight Dynamics*, Darmstadt, Germany, 1986, pp. 65–70.

[30] Howell, K., and Keeter, T., "Station-Keeping Strategies for Libration Point Orbits: Target Point and Floquet Mode Approaches," *AAS/AIAA Spaceflight Mechanics Meeting*, AAS/AIAA, Albuquerque, New Mexico, 1995.

[31] Shirobokov, M., Trofimov, S., and Ovchinnikov, M., "Survey of Station-Keeping Techniques for Libration Point Orbits," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 5, 2017, pp. 1085–1105.

[32] Muralidharan, V., "Stretching Directions in Cislunar Space: Stationkeeping and an Application to Transfer Trajectory Design," Ph.D. dissertation, Purdue University, 2021.

[33] Folta, D., and Vaughn, F., "A Survey of Earth-Moon Libration Orbits: Stationkeeping Strategies and Intra-Orbit Transfers," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Providence, Rhode Island, 2004. https://doi.org/10.2514/6.2004-4741.

[34] Davis, D., Bhatt, S., Howell, K., Jang, J.-W., Whitley, R., Clark, F., Guzzetti, D., Zimovan, E., and Barton, G., "Orbit Maintenance and Navigation of Human Spacecraft at Cislunar Near Rectilinear Halo Orbits," *27th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Society, San Antonio, Texas, 2017.

[35] Newman, C. P., Davis, D. C., Whitley, R. J., Guinn, J. R., and Ryne, M. S., "Stationkeeping, Orbit Determination, and Attitude Control for Spacecraft in Near Rectilinear Halo Orbits," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Snowbird, Utah, 2018, pp. 1–20.

[36] Roberts, C. E., "Long Term Missions at the Sun-Earth Libration Point L1: ACE, SOHO, and WIND," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Girdwood, Alaska, 2011.

[37] Lo, M. W., Williams, B. G., Bollman, W. E., Han, D., Hahn, Y., Bell, J. L., Hirst, E. A., Corwin, R. A., Hong, P. E., Howell, K. C., Barden, B., and Wilson, R., "Genesis Mission Design," *The Journal of the Astronautical Sciences*, Vol. 49, No. 1, 2001, pp. 169–184.

[38] Folta, D., Woodard, M., and Cosgrove, D., "Stationkeeping of the First Earth-Moon Libration Orbiters: The ARTEMIS Mission," *AAS/AIAA Astrodynamics Specialist Conference*, American Astronautical Society, Girdwood, Alaska, 2011.

[39] Muralidharan, V., and Howell, K. C., "Leveraging stretching directions for stationkeeping in Earth-Moon halo orbits," *Advances in Space Research*, 2021. https://doi.org/https://doi.org/10.1016/j.asr.2021.10.028, available online 22 October 2021.

[40] Fujimoto, S., van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," *35th International Conference on Machine Learning (ICML)*, 2018. URL https://arxiv.org/pdf/1802.09477.pdf.

[41] Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, 2nd ed., Springer Series in Statistics, Springer, 2009.

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

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

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

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

[46] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K., "Asynchronous Methods for Deep Reinforcement Learning," *CoRR*, Vol. abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

[47] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," , 2015. URL https://arxiv.org/pdf/1509.02971.pdf.

[48] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 80, edited by J. Dy and A. Krause, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, pp. 1861–1870. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

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

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

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

[52] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, 2015.

[53] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M., "Deterministic Policy Gradient Algorithms," *Proceedings of the 31st International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 32, edited by E. P. Xing and T. Jebara, PMLR, Bejing, China, 2014, pp. 387–395. URL https://proceedings.mlr.press/v32/silver14.html.

[54] van Hasselt, H., Guez, A., and Silver, D., "Deep Reinforcement Learning with Double Q-learning," *CoRR*, Vol. abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

[55] Achiam, J., "Spinning Up in Deep Reinforcement Learning," 2018. URL https://spinningup.openai.com/.

[56] Adams, A., and Vamplew, P., "Encoding and Decoding Cyclic Data," *The South Pacific Journal of Natural Science*, Vol. 16, 1998, pp. 54–58.

[57] Blazquez, E., Beauregard, L., Lizy-Destrez, S., Ankersen, F., and Capolupo, F., "Rendezvous design in a cislunar near rectilinear Halo orbit," *Aeronautical journal*, Vol. 124, No. 1276, 2020, pp. 821–837.

[58] Williams, K., Barden, B., Howell, K., Lo, M., and Wilson, R., "GENESIS Halo Orbit Stationkeeping Design," *International Symposium: Space Flight Dynamics*, Biarritz, France, 2000.

[59] 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.