AUTONOMOUS GUIDANCE FOR MULTI-BODY ORBIT TRANSFERS USING REINFORCEMENT LEARNING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Nicholas B. LaFarge

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Aeronautics and Astronautics

May 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

The Road goes ever on and on
Down from the door where it began.
Now far ahead the Road has gone,
And I must follow, if I can,
Pursuing it with eager feet,
Until it joins some larger way
Where many paths and errands meet.
And whither then? I cannot say.

J.R.R. Tolkein, *The Fellowship of the Ring*

ACKNOWLEDGMENTS

In 2016, seemingly overnight, I decided to change careers, move across the country, and pursue a graduate education in aerospace engineering. All of us in this field understand the excitement and intrigue that space exploration sparks, but for those on the outside, I'm sure our enthusiasm can appear, at best, irrational. For your unwavering support as I dove headfirst into this change, I owe an immense debt of gratitude to my parents, Eleanor and Gedeon. Your encouragement has been appreciated beyond measure. I am incredibly fortunate to have you both as parents.

To my wife, Liene: *Paldies* for all your advice and encouragement as I've navigated through the murky waters of academia. However, more than just your helpful suggestions about succeeding in academic research, thank you for keeping me sane and grounded over the past four years. Thank you for celebrating my triumphs and thank you for all the times you patiently listened when things were not going so well. I am incredibly grateful for all you do for me. Also, acknowledging our family in Indiana would be incomplete without recognizing the contributions of our little furry friends: Roxy and Sherpa. Although you have been more distraction than asset in writing my thesis, you are both great comforts, and easily deserve a spot in any list of acknowledgements.

For my sister, Meg: I remember you once asked me how I make my mind up so easily about big decisions and constantly jump into new endeavors. You claim that this is a quality you lack, yet every time I have taken some new path, you have unblinkingly supported me. You have never counseled me with caution or hesitance. I think the reality is that I have always done what felt right to me, and one reason I can bypass trepidation is because I know I have your love and support. Thank you for seeing me through this far and thank you in advance for seeing me through the rest of the way.

Perhaps my largest debt gratitude is owed to my advisor: Professor Kathleen C. Howell. Professor Howell, despite my unusual background, you saw potential in me, and for that I am immensely grateful. This research came together thanks to your guidance, suggestions, insight, and support. Beyond academics, I am beyond thankful for your support in my professional development. Thank you for introducing me to NASA, encouraging me to pursue Pathways, and supporting me through the fellowship application process. I would also like to thank my committee members, Professor James M. Longuski and Professor David A. Spencer. I appreciate your feedback and your generous use of time in reviewing this document.

Thank you to my research group: while graduate school can often be tumultuous, your company has always felt like an oasis. I appreciate your helpful suggestions in research meetings, your sound advice, and your willingness to always stop what you are doing to help me. Above all, I am thankful for your friendship. Thank you to my research collaborators: Daniel Miller and Professor Richard Linares. I feel fortunate for the opportunity to have worked with you both over the past couple years. I have learned a lot about machine learning from you, and I look forward to future collaboration.

I am extremely grateful to all those at NASA who supported me thus far. At JSC, thank you to the Pathways program and, in particular, thank you to Chris D'Souza, Jack Brazzel, Shane Robinson, Pete Spehar, and James McCabe. I have learned so much from you all, and I look forward to future tours. At JPL, thank you to Mar Vaquero and Juan Senent for enabling my first experience at NASA, and teaching me many practical lessons. Finally, thank you to Dave Folta for your helpful insights, and for your support thus far with NSTRF research.

I have many outstanding friends to thank for all their support along the way. Hank, Erik, Connor, and Fergus: thank you for decades of enduring friendship that has been unshaken by moves and time-zones. Dane, TJ, Mitch, and Ken, thank you for the much-needed distractions, it has been truly “rad”.

I would finally like to thank and acknowledge all those who financially supported my studies. Thank you to the Purdue School of Aeronautics and Astronautics, the Truitt grant, Dr. Waterloo Tsutsui, and the NASA Space Technology Research Fellowship program (NASA grant 80NSSC19K1175). Finally, I acknowledge MIT Supercloud and MIT Lincoln Labatory Supercomputing Centers for providing HPC resources that have contributed to the research results reported within this document.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

LaFarge, Nicholas B. M.S., Purdue University, May 2020. Autonomous Guidance for Multi-Body Orbit Transfers using Reinforcement Learning. Major Professor: Kathleen C. Howell.

While human presence in cislunar space continues to expand, so too does the demand for ‘lightweight’ automated on-board processes. In nonlinear dynamical environments, computationally efficient guidance strategies are challenging. Many traditional approaches rely on either simplifying assumptions in the dynamical model or on abundant computational resources. This research employs reinforcement learning, a subset of machine learning, to produce a controller that is suitable for on-board low-thrust guidance in challenging dynamical regions of space. The proposed controller functions without knowledge of the simplifications and assumptions of the dynamical model, and direct interaction with the nonlinear equations of motion creates a flexible learning scheme that is not limited to a single force model. The learning process leverages high-performance computing to train a closed-loop neural network controller. This controller may be employed on-board, and autonomously generates low-thrust control profiles in real-time without imposing a heavy workload on a flight computer. Control feasibility is demonstrated through sample transfers between Lyapunov orbits in the Earth-Moon system. The sample low-thrust controller exhibits remarkable robustness to perturbations and generalizes effectively to nearby motion. Effective guidance in sample scenarios suggests extendibility of the learning framework to higher-fidelity domains.

1. INTRODUCTION

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

1.1 Problem Definition

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

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

Typically, in the pre-flight trajectory design process, the goal is the construction of an optimal trajectory and a control history that meets mission criteria for propellant usage and time of flight. However, within the context of flight software, optimality is considered secondary to feasibility [7]; the capability to rapidly re-compute a reference path and a feasible control history in-flight is critical for autonomous guidance. By approaching on-board guidance from a machine learning perspective, a closed-loop neural network controller offers the ability to quickly and autonomously compute a control history for a spacecraft with basic linear algebra operations and without iteration. In addition, RL separates the computationally expensive pre-flight learning from the resulting on-board controller and, thus, leverages modern computer hardware while still producing a controller sufficiently lightweight for use in flight.

1.2 Document Overview

This work develops a reinforcement learning approach to spacecraft guidance in the circular restricted three-body problem. The reinforcement learning methodology and dynamical model formulation are developed into a learning framework and demonstrated through a libration point transfer mission application.

- **Chapter 2**: The foundational theory for the Reinforcement Learning (RL) approach in this investigation is presented. This chapter offers an overview of machine learning techniques and neural networks, and discusses prior contributions in astrodynamics. The RL method employed throughout this investigation, Proximal Policy Optimization (PPO), is developed from fundamental reinforcement learning concepts. Policy gradient methods, actor-critic approaches, and trust region policy optimization are discussed as foundational concepts in the development of PPO.
- **Chapter 3**: The dynamical model employed in the reinforcement learning framework is summarized. The equations of motion for the low-thrust circular restricted three-body problem are derived from Newtonian and Hamiltonian mechanics with discussion of the underlying assumptions of the model. Insights into the dynamical model are developed via the Jacobi constant for integration, equilibrium solutions, and zero-velocity surfaces.
- **Chapter 4**: The reinforcement learning environment employed in this investigation is formulated. Reinforcement learning signals, i.e., state, action, and

reward, are detailed. In particular, the reward function is defined and analyzed. The 'episodes' and processes for training are outlined with a discussion of the perturbation model and implementation details.

- **Chapter 5**: The performance of the RL technique is demonstrated through a libration point transfer mission application. Lyapunov-to-Lyapunov heteroclinic transfers serve as reference trajectories; the training process for producing a closed-loop guidance controller is investigated. The performance of the resulting deterministic controller is demonstrated with varying degrees of perturbation. The controller's ability to generalize is exhibited through other transfer scenarios. Algorithm limitations in the sample application are discussed.

- **Chapter 6**: A summary of the research is presented, and suggestions for future work are offered.

2. BACKGROUND: REINFORCEMENT LEARNING

Machine Learning (ML) broadly refers to a class of data science techniques that involve algorithms leveraging predictive and task-performing abilities without direct of explicit instructions. Such approaches are closely related to statistics, and typically involve inferences based on large quantities of data. In many cases, ML tools seek to emulate neurological and biological processes, thus, ML is frequently viewed as a subset of Artificial Intelligence (AI). This distinction between ML and AI is often debated. While ML certainly benefits from contributions in cognitive neuroscience, particularly in the case of artificial Neural Networks (NNs), little progress is evident in understanding “intelligence” as an abstract concept. Current ML techniques focus on replicating low-level decision processes in humans and animals, such as a dog learning to fetch a ball, or a baby learning to take its first steps. Under its original definition, “AI” refers to computer chips replicating the high-level reasoning and intelligence capability of humans. To this end, little, if any, progress has been accomplished in the sixty years since the term first emerged. Due to the distinction between high and low level cognitive functionality, the definition of ML by Professor Michael I. Jordan is representative. Professor Jordan is a machine learning researcher at the University of California Berkeley and defines ML as an umbrella term that includes a variety of tools within data science; AI is viewed as its own distinct field of study. While this distinction may appear pedantic, Professor Jordan pointedly observes,

> We need to realize that the current public dialog on AI – which focuses on a narrow subset of industry and a narrow subset of academia – risks blinding us to the challenges and opportunities that are presented by the full scope of AI [8]

To this end, the ML tools in this investigation are statistical and incorporate optimization techniques rather than emulations of truly intelligent systems.

2.1 Types of Machine Learning

Machine learning is broadly classified into three types: *supervised learning*, *unsupervised learning*, and *Reinforcement Learning* (RL). Supervised learning is the most commonly employed technique in machine learning, where "supervision" refers to the process of tagging data in a training set. In a classic example, a data set contains images of hand-written letters, and "tagging" refers to manually inputting which letter is observed in each image. The learning process then involves accessing this known data and generating functions to perform classification or regression tasks. In the handwriting example, classification corresponds to handwriting recognition and regression corresponds to handwriting generation. A supervised learning approach enables an algorithm to extrapolate to new data by leveraging inferred patterns. Common examples of supervised learning include voice recognition, image classification, product recommendations, personal assistants, and text prediction. While techniques under the supervised learning umbrella perform exceedingly well for many tasks, they are limited by the necessity of large amounts of representative tagged data.

In contrast to supervised learning, unsupervised learning techniques uncover underlying patterns and structure within unlabeled data. Clustering problems are a common example of unsupervised learning; such algorithms seek to group elements together based on some underlying patterns in the data. For example, marketing data companies leverage unsupervised learning to segment consumers into distinct clusters and then introduce targeted advertising to specific groups. Other applications, beyond clustering, include anomaly detection and autoencoding. Unsupervised learning excels in problems with large amounts of a data but with no clear manner of discerning or classifying the data. With an unsupervised learning algorithm, this large amount of data is separated based on auto-computed probability densities that measure the likeliness of subsets of the data belonging to the same grouping. Due to this ability, unsupervised learning is frequently denoted "self-organization".

The final classification of machine learning techniques is *reinforcement learning*. Similar to some supervised learning approaches, RL also seeks to train a decision-making agent. The key difference between supervised and reinforcement learning approaches in decision-making problems is the existence of tagged data. In the case of supervised learning, a tagged data set is created before training, and this knowledge of desired outcomes produces the resulting agent. In contrast, RL requires no such *a-priori* knowledge, and 'learns' by directly interacting with an environment. The notion of learning via environmental interaction is most easily observed in child development and, indeed, many foundational concepts in RL stem from examining the learning process of children and animals. A child is never explicitly told how to walk, but, through a complex set of trial-and-error experiences, discovers the precise combination of motor functions to enable their first steps. Learning such decision-making processes frequently involves both immediate and delayed rewards. These notions of learning-by-interaction, and delayed reward, are the two most important features that distinguish reinforcement learning from other techniques [3].

2.2 Machine Learning Contributions in Astrodynamics

Researchers have recently applied various types of machine learning techniques to common problems in astrodynamics. Studies range from preliminary investigations to demonstrated on-board capability. For example, a basic machine learning algorithm is implemented on-board Mars rovers to aid in identifying geologic features of interest. The Autonomous Exploration for Gathering Increased Science (AEGIS) algorithm enables autonomous targeting on-board Spirit, Opportunity, and Curiosity [9, 10]. On Curiosity's main flight computer, the pointing refinement process requires between 94 and 96 seconds [10] – substantially less than the time necessary to send images to Earth and wait for scientists to manually select features. With on-board computational capability increasing, more complex ML algorithms offer the potential for significant contributions to future autonomous missions. One challenge in establishing ML-driven frameworks is identifying the machine learning techniques most applicable to specific problems in astrodynamics. Understanding and reviewing previous contributions offers insight into potential next steps.

In recent years, progress in applying supervised learning techniques toward astrodynamics problems is increasing rapidly. In trajectory design, regression tasks are typically the most productive. Dachwald applied a shallow Neural Network (NN) to low-thrust trajectory optimization as early as 2004 [11]. More recently, De Smet and Scheeres employ a NN to identify heteroclinic connections in the CR3BP [12]; Parrish and Scheeres explore many nearby optimized trajectories to train a NN to generate low-thrust trajectory corrections [13]. Furfaro et al. explore supervised learning to model a fuel-optimal lunar landing control history [14], and Das-Stuart, Howell, and Folta employ supervised learning to detect types of motion and leverage this knowledge to influence trajectory design [15]. Supervised learning techniques can produce desirable outcomes but carry several notable drawbacks. First, such approaches assume knowledge of the decision-making process. By selecting desired outcomes, the user assumes *a-priori* knowledge of the basis for decision making. This assumption implies that 1) the user-generated data accurately reflects the desired outcomes; and 2) techniques are available to solve the current problem and to generate data. In regimes where such knowledge is absent, supervised learning techniques are not applicable.

Researchers also seek to apply advancements in neural network-based supervised learning image classification to problems in astrodynamics. Two notable applications are terrain relative navigation and pose (i.e., relative position and attitude) estimation. On-board terrain relative navigation remains a challenging task, and neural networks are demonstrated as potentially useful tools for alleviating computational burden. In particular, the abundance of craters on the Moon and on Mars offer potential opportunities for relative navigation. Downes, Steiner, and How [16] and Benedix et al. [17] employ supervised learning methodologies to train neural networks for identifying craters on the Moon and on Mars, respectively. Similarly, numerous authors employ image recognition approaches for estimating the pose of uncooperative spacecraft [18,19]. These investigations involve mixtures of real, augmented, and synthetic imagery. While the results are promising, it is difficult to determine if the training imagery in these investigations is representative of actual inflight scenarios. Different cameras, harsh lighting conditions, and reflective components are among the many challenges faced in supervised learning optical techniques. Furthermore, the inability to easily generate new satellite-based imagery presents serious difficulties in verifying and implementing these techniques.

There are relatively few investigations involving unsupervised learning in astrodynamics. Several authors apply clustering approaches to Poincaré maps. Vaquero and Senent [20], and Pritchett, Howell, and Folta [21] leverage KD-tree structures to locate intersections between manifolds on Poincaré maps. This approach employs unsupervised learning to perform computationally efficient nearest neighbor searches between two discrete sets. In contrast, several authors implement clustering algorithms to organize and uncover data. Nakhijiri and Villac [22] and Villac, Anderson, and Pini [23] employ k-means clustering for stability detection within maps and for classifying periodic orbits around small bodies, respectively. Bosanac [24], as well as Bonasera and Bosanac [25], leverage unsupervised learning to uncover hidden structures within Poincaré maps in multi-body dynamical regimes. These clustering approaches closely align with the goal of unsupervised learning and highlight promising directions for pattern recognition research in problems where the dimensions outweigh the human ability to observe connections within the data. Finally, Gao et al. apply unsupervised learning to spacecraft anomaly detection [26].

Research in applying reinforcement learning techniques to astrodynamics problems has sharply increased in recent years. Investigations are largely concerned with path-finding, landing, small body operations, and multi-body guidance. Other applications include rendezvous guidance in a cluttered environment [27], angle-only intercept guidance [28], stationkeeping in a multi-body environment [29], and detection avoidance [30,31]. For path-finding, in 2017, Das-Stuart, Howell, and Folta leverage Q-learning in conjunction with accessible regions to compute initial guesses for low-thrust transfers in the CR3BP [32]. Following this effort, in 2019, Das-Stuart, Howell, and Folta furthered the investigation by leveraging supervised learning to influence the resulting solution geometry [15], and applied the framework to contingency planning [33]. In 2019, Miller and Linares first employed Proximal Policy Optimization (PPO) for low-thrust trajectory optimization in a multi-body regime [34]. The same year, Miller et al. further demonstrated PPO's decision-making capability for interplanetary low-thrust transfers [35]. In landing problems, Furfaro et al. employ a ZEM/ZEV feedback approach to lunar lander guidance [36], Gaudet and Linares leverage PPO for optimal control landing guidance [37] and for six degree-of-freedom planetary landing [38], and Scorsoglio et al. apply image-based PPO to lunar landing [39]. Gaudet, Linares, and Furfaro also employ PPO to problems associated with Guidance Navigation and Control (GN&C) for asteroid proximity operations, first proposing an RL-based adaptive GN&C framework [40], and then leveraging this framework to perform six degree-of-freedom hovering around an asteroid [41].

Several authors expanded previous contributions to apply RL to guidance problems in a multi-body dynamical regime. In 2020, Sullivan and Bosanac extended Miller's work [34] by leveraging PPO to guide a spacecraft approaching a Lyapunov orbit in the Earth-Moon system [42]. Similarly, LaFarge et al. employ PPO for guidance along multiple heteroclinic transfers between Lyapunov orbits in the Earth-Moon system [43]. This investigation is closely aligned with [43], and provides more context, background, and analysis for the previous results.

2.3 Neural Networks

A Neural Network (NN) is a class of nonlinear statistical models that are frequently employed in ML classification and regression tasks [44]. The term *neural network* encompasses many different types of statistical models with various levels of complexity. When applied correctly, NNs perform exceedingly well, and are a driving factor in ML advancements in the past decade. However, a certain amount of mystery surrounds neural networks and leads many to erroneously view them as "magical" black boxes. While the details are sometimes complex, neural networks are, in fact, similar to other statistical learning methods. While investigations employing NNs are achieving state-of-the-art performance in many domains in recent years, it is important to view them as statistical models. Simply stated, NNs perform well on some tasks, fall short on others and, in general, are not catch-all tools that easily approximate any given function.

While frequently utilized 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. This drawback is commonly labelled the "curse of dimensionality" [3]. Leveraging NNs allows modern algorithms to both access continuous state and action spaces and to easily incorporate additional dimensions.

2.3.1 Neural Network Evaluation

To illustrate functionality, a simple NN is depicted in Figure 2.1. This network employs two scalar inputs, processes them through a single two-node hidden layer, and outputs a scalar value. The depicted network is *feedforward*, that is, there are no cycles between nodes. In other words, values only pass from left-to-right as data is processed through the network. Understanding the evaluation process of NNs informs their suitability for on-board use.

Evaluating a feedforward neural network consists of straightforward linear algebra operations, with several nonlinear element-wise activation functions. After the input layer, each node is computed as a linear combination of weighted values and a bias, and processed through an activation function to incorporate nonlinearity into the model. Hence, in Figure 2.1, the lines connecting nodes each carry an associated weight and each node (except input variables) possesses an associated bias. The weights signify the impact that particular nodes exert on each node in the next layer. The bias allows

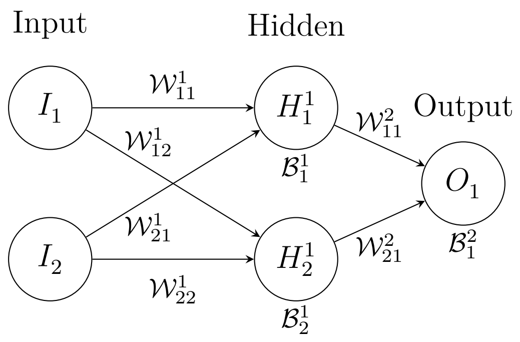


Figure 2.1. Basic feedforward neural network consisting of two inputs, two hidden nodes, and one output variable.

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 of their value is possible. Hence, the weights are typically initialized with random values between 0 and 1, and the biases are initialized at zero. Techniques where guesses for weights and biases are generated from another trained network fall under the umbrella of *transfer learning*.

Consider the two nodes of the hidden layer in Figure 2.1, denoted H_1^1 and H_2^1, where the subscript denotes the index of the node and the superscript denotes the hidden layer to which the node belongs. The values in the hidden layer are computed as,

$$H_1^1 = \alpha^1(I_1\mathcal{W}_{11}^1 + I_2\mathcal{W}_{21}^1 + \mathcal{B}_1^1) \tag{2.1}$$

$$H_2^1 = \alpha^1(I_1\mathcal{W}_{12}^1 + I_2\mathcal{W}_{22}^1 + \mathcal{B}_2^1) \tag{2.2}$$

where α^1 is the activation function employed for the H^1 layer, $\mathcal{B}_j^1$ is the bias term for H_j^1, and $\mathcal{W}_{ij}$ is the weight connecting node I_i in the input layer to node H_j^1 in the hidden layer. The values of H_1^1 and H_2^1 are clearly simple linear combinations of weighted inputs, with the activation function applied to the total sum. Evaluating the two nodes together is expressed more concisely in matrix form,

$$H^1_{2\times1} = \alpha^1 \left(\mathcal{W}^1_{2\times2} I_{2\times1} + \mathcal{B}^1_{2\times1}\right) \tag{2.3}$$

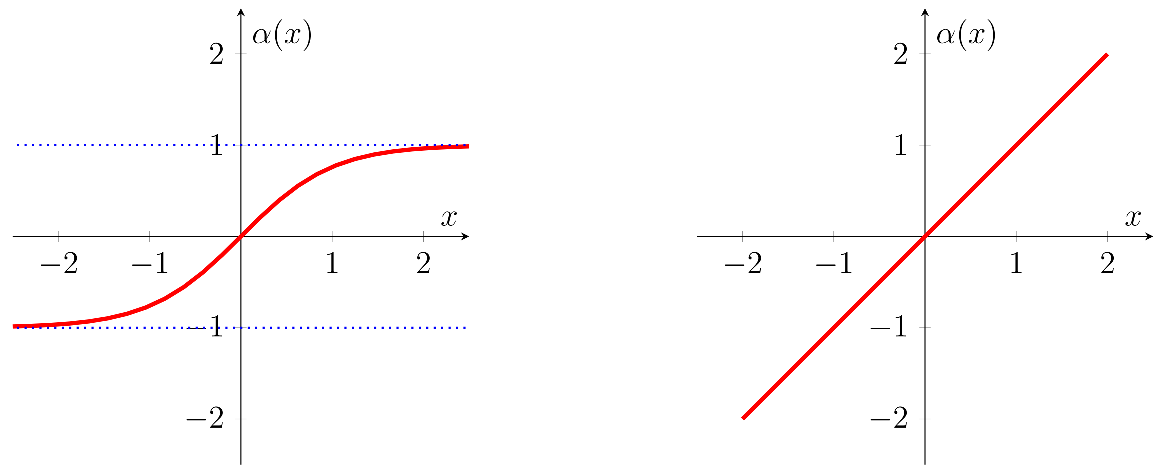
The output is easily computed in the same manner, with inputs received from the hidden layer,

$$O_1 = \alpha^2 \left(\mathcal{W}^2_{1\times2} H^1_{2\times1} + \mathcal{B}^2_1\right) \tag{2.4}$$

Equations (2.3) and (2.4) are the only two operations necessary to evaluate the depicted neural network. In these expressions, the only two potentially nonlinear operations are the activation functions, α^1 and α^2. 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 the network. For this reason, common activation functions include sigmoid, hyperbolic tangent (tanh) and the Rectified Linear Unit (ReLU). This investigation employs tanh to incorporate nonlinearity. Linear activation is also, at times, advantageous. Within RL, networks that produce a single scalar value output frequently employ a linear activation in the output layer. Together, tanh and linear are the only two activation functions employed in this investigation and are plotted in Figure 2.2.

2.3.2 On-board Considerations

Neural networks are potentially well-suited for use on a flight computer. While training is computationally complex, the evaluation process is relatively simple; several matrix multiplications and additions, combined with element-wise activation functions, encompass the entire process. Furthermore, flight software often requires algorithms to predictably complete in a given number of CPU cycles. For example, Orion orbit guidance is required to complete in a fixed number of steps, which poses significant difficulty in numerical integration and targeting processes [45]. If



(a) Hyperbolic tangent (tanh)

(b) Linear

Figure 2.2. Neural network activation functions employed in this investigation.

a function is predictable, a certain amount of processing time is easily allocated for it. 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 accumulation operations" and a "substantial amount of memory to store data" [46]. Flight hardware to address these difficulties is currently being developed. NASA is actively soliciting proposals for "neuromorphic" processors to enable in-space autonomy [46]. These specialized processors, inspired by the human brain [47], allow for dedicated low-power NN evaluations in space. Adoption of neuromorphic hardware into flight systems will render machine learning approaches more accessible and productive. Neuromorphic computing could enable efficient autonomous control, decision making, and on-board adaptive learning [48].

2.4 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 cause the learning process to cease. In these cases, the environment typically resets to an initial configuration, and the process begins anew. These are identified as *episodic* tasks, where each episode signifies an attempt by the agent to solve a problem. A schematic for the high-level process of an RL agent is depicted in Figure 2.3. This diagram highlights the three signals that enable communication between the agent and environment at time steps t and $t+1$: state, action, and reward. While the RL literature prefers the terms *agent*, *environment*, and *reward*, these expressions are analogous to the more common engineering terms *controller*, *controlled system* (or *plant*), and *control signal* [3].

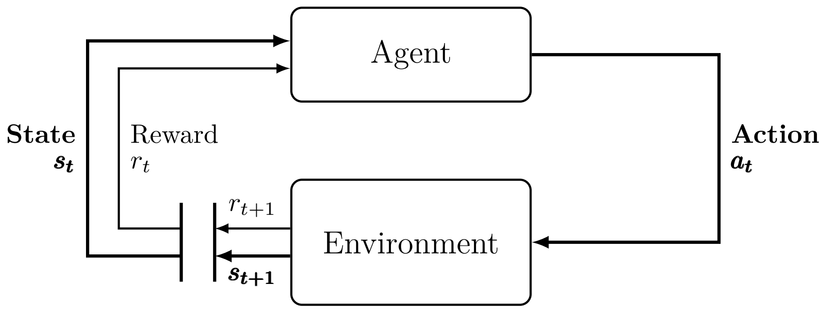


Figure 2.3. The agent-environment process in a Markov decision process (reproduced from Sutton and Barto, Ref [3], p.48. Figure 3.1).

To clarify the RL learning process and terminology, consider the example of a person learning to play chess. Assuming they know the potential movements for each piece, the person (or *agent*), plays their first game. At any given point, the board (the *environment*) is at some specified configuration. The agent visually observing the location of each piece communicates the current *state* of the board, so that the agent understands the available moves (or *actions*). In their first game, the agent possesses no notion of strategy and, therefore, moves randomly. The means by which the player selects their move is their *policy.* In chess, both immediate and delayed *rewards* inform the agent concerning the suitability of particular moves. For example, capturing a piece is an example of an immediate reward, while winning in 4 moves is a delayed reward. Over time, the agent improves at chess by learning to maximize the expected future reward. For humans, mastering chess involves non-intuitive decision-making that incorporates complex pattern recognition and the identification of future rewards. Each time a new game (or *episode*) starts, the player applies lessons from the previous game to inform future play. Assuming no exterior interaction (the agent is not allowed to consult an instructional chess book), the learning process occurs over many thousands of episodes, after which an expert chess player is produced.

Traditional computer chess methods rely on heuristics to build large tree structures, which are pruned, searched, and evaluated to produce quality moves. The strongest such engine, Stockfish, championed 8 sequential Chess.com tournaments beginning in 2011. However, recent progress in machine learning produced an RL-driven chess engine that outperforms traditional state-of-the-art techniques. In 2019 and 2020, Leela Chess Zero (LCZero) narrowly defeated Stockfish to become the top computer chess engine [49, 50]. Leela Chess Zero is an RL-based framework that builds off success with AlphaGo Zero [51]. Training with only rule knowledge, LCZero improves over 300 million games of self-play. The success of LCZero demonstrates the ability of RL to outperform traditional methods in challenging problems.

The fundamental notions in the chess example form the basis for training an artificial agent using RL. 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 [3]. 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 $< \mathcal{S}, \mathcal{A}, \mathcal{P}, r, q_0, \gamma >$, where $\mathcal{S}$ and $\mathcal{A}$ are the sets of all possible states and actions, respectively, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the state-transition probability distribution, $r : \mathcal{S} \rightarrow \mathbb{R}$ is the reward function, q_0 is the distribution of the initial states $\boldsymbol{s_0}$, and $\gamma \in [0, 1]$ is the discount factor [52].

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 [3]. 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 be represented as the state $\boldsymbol{s_t}$, and no such distinction is necessary.

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

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \tag{2.5}$$

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

$$V_\pi(\boldsymbol{s_t}) = \mathbb{E}_\pi\left[G_t\right] = \mathbb{E}_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r(\boldsymbol{s_k})\right] \tag{2.6}$$

where, assuming the agent follows policy π, $\mathbb{E}_\pi\left[G_t\right]$ is the expected value of a random variable at any time step t. It is similarly useful to estimate the value of a state-action pair. The *state-action value function*, denoted $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$, computes the value $V_\pi(\boldsymbol{s_t})$ of taking action $\boldsymbol{a_t}$ at $\boldsymbol{s_t}$, and subsequently following the policy π. This function, $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$, is closely related to the state-value function, Eq. (2.6), and is computed as,

$$Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t}) = \mathbb{E}_\pi\left[G_t\right] = \mathbb{E}_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r(\boldsymbol{s_k}, \boldsymbol{a_k})\right] \tag{2.7}$$

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

$$V_\pi(\boldsymbol{s_t}) = \mathbb{E}_{\boldsymbol{a_t}\sim\pi}\left[Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})\right] \tag{2.8}$$

The functions $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ and $V_\pi(\boldsymbol{s_t})$ differ in the action given that $\boldsymbol{a_t}$ in $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ is not necessarily sampled from policy π. In practice, many modern algorithms do not compute value functions directly but instead estimate them using neural networks.

Instead of directly estimating $V_\pi(\boldsymbol{s_t})$ or $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$, some RL methods, such as the one employed in this research effort, are concerned with a state-action pair's value compared to average. In these approaches, a relative value, termed the *advantage* function, is computed as the difference between the state-action-value function and the state-value function,

$$A_\pi(\boldsymbol{s_t}, \boldsymbol{a_t}) = Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t}) - V_\pi(\boldsymbol{s_t}) \tag{2.9}$$

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

Reinforcement learning algorithms are broadly grouped as *model-free* and *model-based* algorithms. Model-based algorithms allow for learning when the environmental dynamics are unknown. This investigation involves a known dynamical model and, hence, model-based algorithms are not employed. Rather, within model-free algorithms, two main categories exist for reinforcement learning: value and policy optimization methods. Value optimization typically involves learning $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ directly, denoted Q-learning, and constructing a policy π that exploits this knowledge. This learning is frequently accomplished by a process of iteration using the Bellman equation. If the value of state-action pairs is known, then an optimal policy is easily formulated by selecting the action that yields the highest expected return. Classical approaches to Q-learning are formulated by including all possible state/action combinations in a table and learning by visiting all combinations infinitely many times [3]. Modern approaches involve estimating $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$ using a neural network and form the basis for the well-known Deep Q-Network (DQN) proposed by Mnih et al. [53].

Instead of producing an optimal policy based on the estimation of $Q_\pi(\boldsymbol{s_t}, \boldsymbol{a_t})$, policy optimization methods seek to directly learn a parameterized policy, $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$, where $\boldsymbol{\theta}$ represent a policy parameter vector. In contrast to value optimization methods, such as Q-Learning and DQN, policy optimization methods interact 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 to learn both the policy (i.e., *actor*) and the value (i.e., *critic*) functions. Methodologies involving actor and critic networks are of particular recent interest and are employed in this investigation.

A final point that distinguishes RL approaches is *on-policy* vs. *off-policy* algorithms. In off-policy approaches, such as Q-Learning, an optimal policy may be learned directly regardless of the data collection approach. In contrast, policy optimization algorithms typically require data sampled from the most recent version of the policy and are, therefore, termed on-policy methods. This investigation utilizes policy optimization and, therefore, all algorithms are on-policy methods.

2.4.1 Policy Gradient Methods

Policy gradient methods are a subclass of policy optimization algorithms that utilize gradient descent to optimize policy parameters. In these approaches, during training, an action is sampled from a diagonal Gaussian distribution (i.e., a multivariate normal distribution) $\boldsymbol{a_t} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})$, where the mean values are computed by the agent. Including variance in the selected actions during training allows the agent to more thoroughly explore the environment and action spaces. The mean values are typically produced by an 'actor' neural network, that employs a state or observation as input and outputs the mean of each action variable. The actor NN employed in this investigation appears in Figure 2.4. The standard deviation for the resulting distribution is typically computed in one of two ways. First, the standard deviations can be included as outputs from the neural network, so that all values in the actor distribution are state-dependent. Alternatively, a more common approach in recent implementations is maintaining a vector of log standard deviations such that the variance is not a function of the state. The latter technique is employed in this investigation. After training is complete, the standard deviations are no longer incorporated, and the actor neural network is employed directly as a deterministic controller.

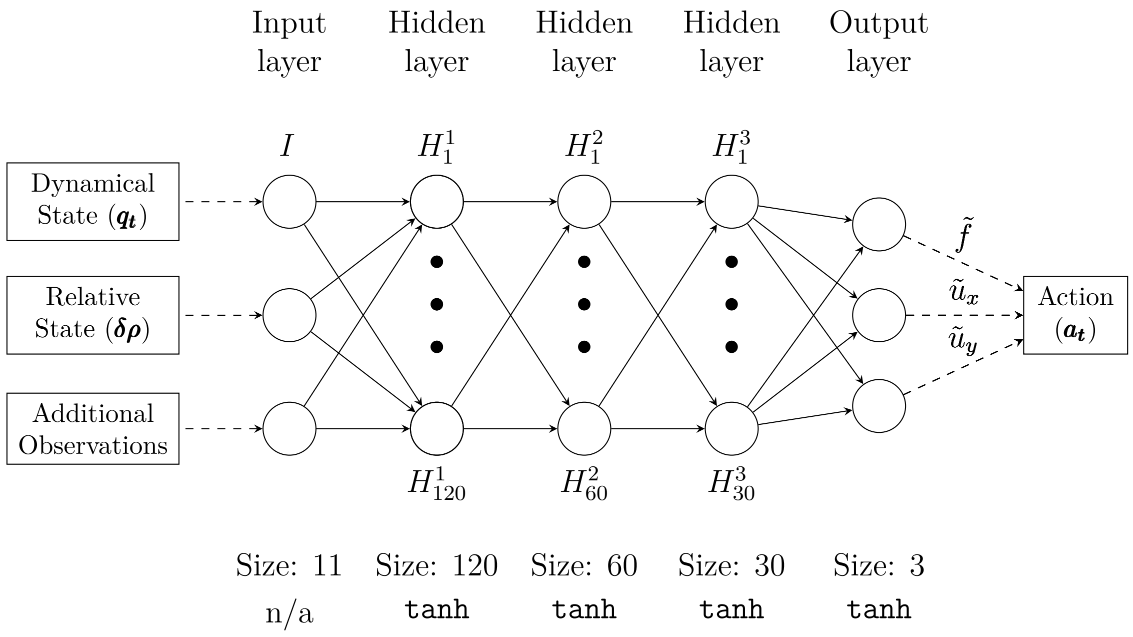


Figure 2.4. Actor neural network employed in the proximal policy optimization algorithm (from LaFarge et al., Ref [43], Fig. 1).

The agent's policy is learned by optimizing the weights of the actor NN, denoted $\boldsymbol{\theta}$, to maximize performance. As detailed by Sutton and Barto [3], performance is quantified by using the maximization of the value function as an objective function,

$$J(\boldsymbol{\theta}) = V_{\pi\boldsymbol{\theta}}(\boldsymbol{s_0}) \tag{2.10}$$

where $\boldsymbol{s_0}$ is a non-random start state. With the objective function in Eq. (2.10), it follows that the update gradient for the actor is given by,

$$g = \nabla_{\boldsymbol{\theta}} \left[V_{\pi\boldsymbol{\theta}}(\boldsymbol{s_0}) \right] \tag{2.11}$$

The gradient in Eq. (2.11) is not typically computed directly. In policy gradient methods, direct computation results in prohibitively noisy gradients that prevent learning. Alternatively, using the critic network in actor-critic methods results in less noisy estimates, but introduces a significant bias into the computation [54]. Therefore, most algorithms estimate the gradient using a methodology detailed by Schulman et al. [54]. A common policy gradient estimator is,

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t}) \hat{A}_t \right] \tag{2.12}$$

with the corresponding policy gradient loss function,

$$L^{PG}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t}) \hat{A}_t \right] \tag{2.13}$$

Many of the earlier policy gradient algorithms directly optimized Eq. (2.13) (e.g., REINFORCE [55]). However, the variance in gradient estimation frequently leads to destructively large policy updates in these strategies. For example, with a stochastic policy, an unexpected experience can yield extremely steep gradients, resulting in updates that render certain actions orders of magnitude more or less probable. Applying such updates destabilizes the learning process and prohibits learning an optimal policy [56].

Repeatedly optimizing Eq. (2.13) leads to potentially destructive policy updates, so Schulman et al. introduce a "surrogate" loss function that avoids large policy updates and produces training that follows monotonic improvement. The resulting algorithm is denoted Trust Region Policy Optimization (TRPO) and forms the basis for many productive modern RL algorithms [52]. Prior to TRPO, gradient-based methodologies never outperformed dynamic programming on continuous control tasks. However, by including limiting bounds on policy updates, TRPO produces powerful and complex policies that outperform dynamic programming for many problems [52]. In TRPO, the agent samples actions from a policy distribution $\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$. With the state-action-reward trajectory generated under $\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$, a new policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})$ is optimized that, in turn, becomes $\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$. By creating two distributions, the change in policy is quantified by employing the Kullback-Leibler (KL) divergence, which measures the similarity between two probability distributions. A constraint is then introduced into the optimization scheme to insert an upper limit on the KL divergence between $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}_{\text{old}}}$, resulting in monotonic policy improvement. Together, the optimization scheme under TRPO becomes,

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[R_t(\boldsymbol{\theta}) \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot|\boldsymbol{s_t}), \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s_t}) \right] \right] \leq \delta \end{aligned} \tag{2.14}$$

where δ is a limit on the change in policy distributions and the ratio,

$$R_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})} \tag{2.15}$$

is the probability of an action, given a certain state under a new optimized policy, divided by the probability of that action under the previous policy distribution. The constraint inequality prevents prohibitively large updates by ensuring that the change in policy distribution remains within a specified bound. The ratio in Eq. (2.15) is most easily understood by considering the relative magnitudes of the numerator and denominator. If the optimization step causes an action $\boldsymbol{a_t}$ to become more probable, then $\pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t}) > \pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a_t}|\boldsymbol{s_t})$, and so $R_t(\boldsymbol{\theta}) > 1$. Conversely, if the action becomes less probable, then $R_t(\boldsymbol{\theta}) < 1$.

While TRPO is successful in many challenging domains, there are several noted limitations. First, the algorithm itself is very complex and is, therefore, prone to implementation errors. Furthermore, TRPO involves a hard constraint in the optimization process due to the difficulty in selecting a single coefficient that transforms the KL-divergence constraint in Eq. (2.14) into a penalty [56]. Finally, TRPO is not compatible with tasks that include noise or parameter sharing.

2.4.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a simpler and more flexible alternative to achieve the same high performance as TRPO [56]. In formulating PPO, the TRPO KL divergence constraint, Eq. (2.14), is replaced with a clipping factor that controls the maximum allowable update. The change in probability distribution is quantified by ratio in Eq. (2.15). By introducing clipping, PPO optimizes a surrogate objective, that is,

$$L^{CLIP}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min\left(R_t(\boldsymbol{\theta})\hat{A}_t, \operatorname{clip}(R_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{2.16}$$

where $R_t(\boldsymbol{\theta})$ is the probability ratio defined in Eq. (2.15). Multiplying $R_t(\boldsymbol{\theta})$ by the estimate of the advantage function forms the core of PPO: $R_t(\boldsymbol{\theta})\hat{A}_t$. Recall that the ratio $R_t(\boldsymbol{\theta}) > 1$ implies an action is now more probable under a new, optimized probability distribution. Next, the ratio $R_t(\boldsymbol{\theta})$ is multiplied by the advantage function $\hat{A}_t$, defined in Eq. (2.9), which measures of the quality a particular action as compared with the expected value. If the advantage is positive, an action is better than expected, and the optimization scheme seeks to enable this action to be more probable. Conversely, a worse-than-average action becomes less probable. Finally, the minimum value between the unclipped $R_t(\boldsymbol{\theta})\hat{A}_t$ and the clipped objective function between $1 \pm \epsilon$ forms a pessimistic lower bound for the unclipped objective. With $\epsilon = 0.2$ as a suggested value, an action is, at most, 20% more or less probable under PPO's optimization scheme. Analogous to the constraint in TRPO, clipping is included in PPO to eliminate the destructively large policy updates that are common in vanilla policy gradient schemes.

An alternative approach to PPO, rather than clipping the surrogate objective function, is to instead introduce an adaptive KL penalty coefficient [56]. The adaptive penalty approach is a hybrid between PPO and TRPO. Rather than clipping the objective function, the KL divergence is introduced as a penalty,

$$L^{KLPEN}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[R_t(\boldsymbol{\theta})\hat{A}_t - \beta \mathrm{KL}\left[\pi_{\boldsymbol{\theta}_{\mathrm{old}}}(\cdot|\boldsymbol{s_t}), \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s_t})\right]\right] \tag{2.17}$$

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

$$d = \hat{\mathbb{E}}_t \left[\mathrm{KL}\left[\pi_{\boldsymbol{\theta}_{\mathrm{old}}}(\cdot|\boldsymbol{s_t}), \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s_t})\right]\right] \tag{2.18}$$

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

$$\beta = \begin{cases} \beta/2 & \text{if } d < d_{\text{targ}}/1.5 \\ \beta \times 2 & \text{if } d > d_{\text{targ}} \times 1.5 \end{cases} \tag{2.19}$$

while the parameters 1.5 and 2 are heuristically determined. However, Schulman et al. note that the algorithm is not sensitive to the specific choices 1.5 and 2 [56]. The update rule for β is simply understood as follows: when the probability distribution endures more change, the penalty coefficient is increased to encourage smaller policy updates. Conversely, when the distribution is less variable, the penalty is decreased to allow larger updates. While clipped PPO typically performs better than the Kl penalty baseline [56], this research employs a KL penalty approach due to a dramatic increase in performance compared with vanilla-clipped PPO. The specific implementation of PPO in this investigation is based on the open source work of Patrick Coady[1], and includes several other minor customizations to the core algorithm.

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

3. BACKGROUND: DYNAMICAL MODEL

A suitable dynamical model is necessary for evaluating the proposed guidance framework. This investigation demonstrates a neural network controller's ability to function in challenging regions of space. The three-body problem serves a relevant model that is representative of observed natural motion in cislunar space. While the proposed guidance framework does not depend on any particular model, the planar Circular Restricted Three-Body Problem (CR3BP) serves as a useful environment for preliminary evaluation because it both covers challenging regions of space while remaining simple enough for initial analysis of the guidance scheme. Additionally, low-thrust propulsion is included to demonstrate algorithmic performance despite limited control authority.

3.1 The *N*-Body Problem

The motion of a particle governed by the gravitational influence of any number of celestial bodies is derived using Newton's second law. In the most general formulation, particle P_i, with mass M_i, moves relative to an inertially fixed point O in a gravity field consisting of the remaining N-1 bodies. A vector diagram for this dynamical system is depicted in Figure 3.1. The dimensional position vectors $\boldsymbol{R_i}$ are defined with respect to an inertial reference frame $(\hat{X}, \hat{Y}, \hat{Z})$ with origin at base-point O. Here, vector quantities are denoted by bold typeface, while scalar quantities are italic. Capital letters signify dimensional values, where lowercase letters reflect nondimensional values. Assuming that each body is controbaric, the N-problem is mathematically modeled in terms of the second-order vector differential equation,

$$M_i \ddot{\boldsymbol{R}}_{\boldsymbol{i}} = -\tilde{G} \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{M_i M_j}{R_{ji}^3} \boldsymbol{R_{ji}} \tag{3.1}$$

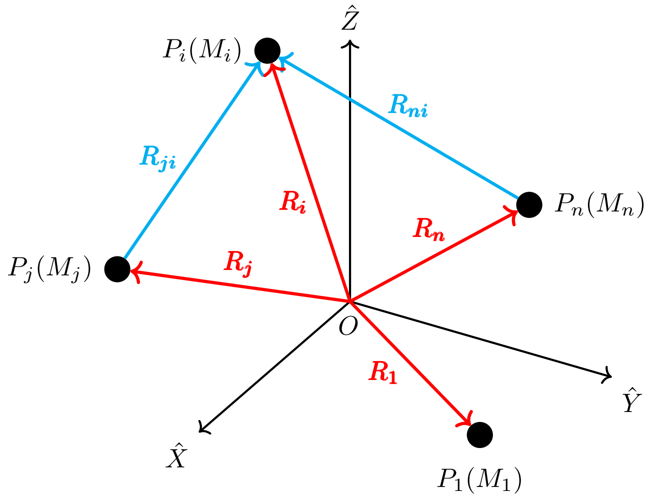


Figure 3.1. Vector definition in the N-Body problem.

where $\tilde{G} \approx 6.67408 \times 10^{-20}$ $[\text{km}^3/(\text{kg} \cdot \text{s}^2)]$ is the universal gravitational constant, M_i and M_j are the mass of P_i and P_j, respectively, and $\boldsymbol{R_{ji}}$ is the position vector from P_j to P_i, with magnitude R_{ji}, depicted in Figure 3.1. The relative distance between two particles is easily computed from the difference in their positions, $\boldsymbol{R_{ji}} = \boldsymbol{R_i} - \boldsymbol{R_j}$. An overdot signifies the time derivative of a quantity as viewed by an inertial observer. Hence, $\ddot{\boldsymbol{R}}_{\boldsymbol{i}}$ represents the acceleration of particle P_i in the inertial frame.

Including all the significant celestial bodies in the equations of motion increases the fidelity of the dynamical model. However, reducing the total number of bodies in the dynamical system enables further insight in the problem. For example, many preliminary designs involve the relative two-body problem, for which Kepler's equation offers a closed-form analytical solution. Keplarian motion is useful in regions of space where the motion of a spacecraft is largely governed by a single gravitational body, such a spacecraft in Low Earth Orbit (LEO). In regimes where two bodies significantly influence the motion of a spacecraft, such as in cislunar space, a third body is frequently included to inform analysis of the multi-body dynamics.

3.2 The Circular Restricted Three-Body Model

In many relevant dynamical environments, the motion of a particle is dominated by the gravitational influence of two spherically symmetric gravity fields. The N-body problem, defined in Eq. (3.1), is reduced to incorporate only three bodies, that is $N = 3$, and,

$$\ddot{\boldsymbol{R}}_{\boldsymbol{3}} = -\tilde{G}\frac{M_1}{R_{13}^3}\boldsymbol{R_{13}} - \tilde{G}\frac{M_2}{R_{23}^3}\boldsymbol{R_{23}} \tag{3.2}$$

where P_1 and P_2 represent gravitational bodies that govern the motion of P_3. Equation (3.2) is the most general form of the problem of three-bodies, however three additional assumptions are included to enable further analysis.

3.2.1 Simplifying Assumptions

The general three-body problem, represented in Eq. (3.2), is focused on the analysis of the mutual gravitational influence of all three bodies. However, in many cases of interest, the mass of one body is much smaller than the other two and, thus, its relative gravitational impact is much less significant, and it is reasonably neglected. For example, the mass of a spacecraft moving with respect to two celestial bodies is neglected due to the orders of magnitude difference between the relative mass of the spacecraft and the celestial bodies. In such a scenario, with the mass of P_3 neglected, the motion of P_1 and P_2 forms an isolated two-body system that is characterized using Keplerian motion. The two massive bodies, P_1 and P_2, form a *primary system*, where both bodies rotate around a common barycenter, B, and P_3 moves with respect to the barycenter in any spatial dimension. Primary systems of interest include sun-planet systems (e.g., Sun-Earth) and planet-moon systems (e.g., Earth-Moon). The following simplifying assumptions yield a problem that is more tractable:

- **Assumption 1**: $M_3 << M_1, M_2$

  - The mass of P_3 is infinitesimal relative to the masses of P_1, P_2. Hence, P_3 does not influence the motion of P_1 and P_2

- **Assumption 2**: P_1 and P_2 represent an isolated two-body conic system
  - The primary system involves closed conic motion (ellipse)
  - By convention, $M_1 > M_2$
  - P_1 and P_2 move in the same plane, but P_3 can move freely through the system

The restricted three-body problem allows for the inclusion of eccentricity in the primary two-body conic system. While eccentric primary systems are useful in higher-fidelity models and for highly elliptic primary systems, many useful applications involve regimes that are nearly circular. A third assumption is summarized as,

- **Assumption 3**: P_1 and P_2 move on circular orbits around B.

Together, Assumptions 1-3 form the basis for the **Circular Restricted Three-Body Problem** (CR3BP). The CR3BP is employed in this investigation as an example of a nonlinear dynamical environment that is relevant to upcoming missions.

3.2.2 Circular Restricted Three-Body Problem Formulation

The CR3BP, a model for motion of an infinitesimal mass moving under the influence of two massive bodies, is employed in this investigation to evaluate a proposed guidance scheme within the context of a complex dynamical regime. In this model, P_1 and P_2 form the primary system as they move in circular orbits about B; P_3 moves freely with respect to the barycenter, as depicted in Figure 3.2. Bodies P_1 and P_2 are located along the rotating $\hat{x}$ axis by the vectors $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$, respectively. The position of P_3 relative to P_1 is written as $\boldsymbol{r_{13}}$, and similarly $\boldsymbol{r_{23}}$ locates P_3 relative to P_2. For convenience, the position and velocity of P_3, denoted $\boldsymbol{r_3}$ and $\boldsymbol{v_3}$, respectively, are propagated in a rotating reference frame. The rotating frame is spanned by dextral

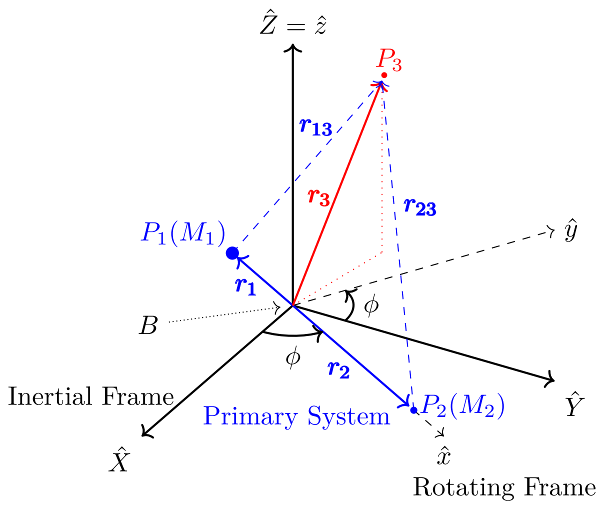


Figure 3.2. CR3BP vector definitions with respect to an inertial frame where $\mu = 0.2$.

orthonormal vectors $\hat{x}$-$\hat{y}$-$\hat{z}$ that rotate with the primary system, denoted by dashed lines in Figure 3.2, such that $\hat{x}$-$\hat{y}$ are oriented by an angle ϕ from the inertial unit vectors $\hat{X}$-$\hat{Y}$. The spatial component, $\hat{Z} = \hat{z}$, is defined as parallel to the orbital angular momentum vector, $\hat{x}$ is oriented from P_1 to P_2, and $\hat{y}$ completes the right-handed triad.

3.2.3 Characteristic Quantities and Nondimensionalization in the Circular Restricted Three-Body Problem

Quantities in the CR3BP are nondimensionalized to reduce error and enable the analysis to be more generalizable. The equations of motion are numerically integrated, with familiar floating point and truncation errors in the integration process. Furthermore, formulating the problem in terms of nondimensional quantities yields additional insight when extrapolating results between multiple CR3BP systems with similar mass ratios. In the CR3BP, the mass ratio between the first and second primary body is defined as,

$$\mu = \frac{M_2}{M_1 + M_2} \tag{3.3}$$

Introducing $\mu \in [0,\, 0.5]$ allows any primary systems with the same mass ratio to be represented by the same equations of motion. Thus, insight from a particular system is more easily applicable to another similar system. For example, analysis of the Saturn-Titan system ($\mu = 2.3664 \cdot 10^{-4}$, nondim) easily transitions into the Neptune-Triton system ($\mu = 2.0882 \cdot 10^{-4}$, nondim) due to their similar mass ratio. To nondimensionalize, characteristic quantities for position, mass, and time are selected arbitrarily but, in general, yield nondimensional values of the same order of magnitude. For this investigation, the characteristic length is defined as the distance between the primary bodies, i.e.,

$$l^* = R_1 + R_2 \tag{3.4}$$

where R_1 and R_2 are the dimensional distances from B to P_1 and P_2, respectively. Next, the characteristic mass equals the sum of the masses of P_1 and P_2,

$$m^* = M_1 + M_2 \tag{3.5}$$

Using the mass ratio μ in Eq. (3.3), the nondimensional masses of P_1 and P_2 are expressed as,

$$\frac{M_1}{m^*} = 1 - \mu \qquad \frac{M_2}{m^*} = \mu \tag{3.6}$$

Furthermore, by defining the origin of the system as the barycenter, the center of mass definition delivers the vector relationship $((-M_1R_1 + M_2R_2)/m^*)\,\hat{x} = \mathbf{0}$. Using the definition of l^* in Eq. (3.4) and the relationships given in Eqs. (3.4) and (3.6), it follows that the nondimensional distances from the barycenter to each primary body are,

$$r_1 = \mu \qquad\qquad r_2 = 1 - \mu \tag{3.7}$$

The nondimensional distances in Eq. (3.7) highlight the relationship between the mass of the primary bodies and their distances to the common barycenter.

Both characteristic quantities l^* and m^* are intuitive and correspond to physical values, whereas the characteristic time, t^*, is selected such that the nondimensional gravitational constant, G, is equal to one. Recall that the units for $\tilde{G}$ are km^3/(kg·s^2). Hence, select t^* such that,

$$G = \frac{\tilde{G}m^*(t^*)^2}{(l^*)^3} = 1 \tag{3.8}$$

Solving for t^* produces the final expression,

$$t^* = \sqrt{\frac{(l^*)^3}{\tilde{G}m^*}} \tag{3.9}$$

Recall that P_1 and P_2 are assumed to be in circular orbits around a common barycenter. The dimensional mean motion is evaluated as $N = \sqrt{\tilde{G}m^*/(l^*)^3}$. It follows that $N = 1/t^*$. Multiplying N by t^*, the nondimensional mean motion is,

$$n = Nt^* = 1 \tag{3.10}$$

Mean motion represents the angular speed necessary to complete one period of an orbit. Noting that $n = 2\pi/\mathbb{P}$, where $\mathbb{P}$ is the orbital period, it follows that the nondimensional period of the primary system is 2π. The three-body system employed in this investigation is the Earth-Moon system. The characteristic quantities are,

$$\mu = 0.012004715741012, \text{ nondim}$$
$$l^* = 384747.962856037, \text{ km}$$
$$t^* = 375727.551633535, \text{ s}$$
$$m^* = 6.045825574495057 \cdot 10^{24}, \text{ kg}$$

For comparison, the characteristic quantities of additional systems are summarized in Table 3.1.

Derivation of the Equations of Motion

The equations of motion that govern P_3 are derived from Newton's Second Law, as detailed in *Principia*. Recall that, for convenience, P_3 is located with respect to B

Table 3.1. Characteristic quantities in various three-body systems.

System	μ, nondim	l^*, km	t^*, days	m^*, kg
Earth-Moon	$1.200472e^{-2}$	$3.847480e^{5}$	4.348699	$6.045826e^{24}$
Sun-Earth	$3.003481e^{-6}$	$1.495979e^{8}$	58.13237	$1.988481e^{30}$
Sun-Jupiter	$9.536839e^{-4}$	$7.781905e^{8}$	$6.893696e^{2}$	$1.990374e^{30}$
Jupiter-Europa	$2.528018e^{-5}$	$6.970048e^{5}$	0.5983692	$1.898235e^{27}$
Saturn-Titan	$2.366393e^{-4}$	$1.221838e^{6}$	2.537796	$5.684706e^{26}$
Mars-Phobos	$1.654872e^{-8}$	$9.373719e^{3}$	$5.075611e^{-2}$	$6.417120e^{23}$
Pluto-Charon	0.1085399	$1.959621e^{4}$	1.016558	$1.461615e^{22}$
Neptune-Triton	$2.088192e^{-4}$	$3.547571e^{5}$	0.9353305	$1.024340e^{26}$

in the rotating frame, as depicted in Figure 3.2. The nondimensional position of P_3 is defined as,

$$\boldsymbol{r_3} = x\,\hat{x} + y\,\hat{y} + z\,\hat{z} \tag{3.11}$$

where $\hat{x}$-$\hat{y}$-$\hat{z}$ are unit vectors fixed in the rotating frame, and x-y-z are nondimensional scalar quantities reflecting the components or measure numbers. To apply Newton's Second Law, the general form of the three-body problem from Eq. (3.2) is first nondimensionalized, resulting in the nondimensional three-body problem,

$$\ddot{\boldsymbol{r}}_{\boldsymbol{3}} = -\frac{1-\mu}{r_{13}^3}\boldsymbol{r_{13}} - \frac{\mu}{r_{23}^3}\boldsymbol{r_{23}} \tag{3.12}$$

where $\ddot{\boldsymbol{r}}_{\boldsymbol{3}} = \ddot{\boldsymbol{R}}_{\boldsymbol{3}} \cdot [(t^*)^2/l^*]$ is the nondimensional acceleration vector, and $\boldsymbol{r_{13}} = \boldsymbol{R_{13}} \cdot [1/l^*]$, $\boldsymbol{r_{23}} = \boldsymbol{R_{23}} \cdot [1/l^*]$ are the nondimensional vectors locating P_3 with respect to P_1 and P_2, respectively. Both $\boldsymbol{r_{13}}$ and $\boldsymbol{r_{23}}$ are simply deduced as,

$$\boldsymbol{r_{13}} = (x - r_1)\hat{x} + y\,\hat{y} + z\,\hat{z} \tag{3.13}$$

$$\boldsymbol{r_{23}} = (x - r_2)\hat{x} + y\,\hat{y} + z\,\hat{z} \tag{3.14}$$

Using Eqs. (3.13) and (3.14), and the relationships in Eq. (3.7), the scalar components of $\ddot{\boldsymbol{r}}_{\mathbf{3}}$ are,

$$\ddot{r}_{3,x} = -\frac{(1-\mu)(x+\mu)}{r_{13}^3} - \frac{\mu(x-1+\mu)}{r_{23}^3} \tag{3.15}$$

$$\ddot{r}_{3,y} = -\frac{(1-\mu)y}{r_{13}^3} - \frac{\mu\, y}{r_{23}^3} \tag{3.16}$$

$$\ddot{r}_{3,z} = -\frac{(1-\mu)z}{r_{13}^3} - \frac{\mu\, z}{r_{23}^3} \tag{3.17}$$

where the magnitudes of $\boldsymbol{r}_{\mathbf{13}}$ and $\boldsymbol{r}_{\mathbf{23}}$ are

$$r_{13} = \sqrt{(x+\mu)^2 + y^2 + z^2} \tag{3.18}$$

$$r_{23} = \sqrt{(x-1+\mu)^2 + y^2 + z^2} \tag{3.19}$$

The acceleration terms $\ddot{r}_{3,x}, \ddot{r}_{3,y}, \ddot{r}_{3,z}$ are computed by differentiating the position vector $\boldsymbol{r}_{\mathbf{3}}$ in the rotating frame. Using the Basic Kinematic Equation (BKE), the kinematic expression velocity is first evaluated,

$$\dot{\boldsymbol{r}}_{\mathbf{3}} = \frac{^{\mathcal{I}}d}{dt}\boldsymbol{r}_{\mathbf{3}} = \frac{^{\mathcal{R}}d}{dt}\boldsymbol{r}_{\mathbf{3}} + \underbrace{^{\mathcal{I}}\boldsymbol{\omega}^{\mathcal{R}}}_{n\hat{z}} \times \boldsymbol{r}_{\mathbf{3}} \tag{3.20}$$

where $\mathcal{I}$ represents the inertial frame, $\mathcal{R}$ represents the rotating frame, and $^{\mathcal{I}}\boldsymbol{\omega}^{\mathcal{R}}$ is the angular velocity of the rotating frame with respect to the inertial frame. For a circular orbit, this angular rate is simply the mean motion of the orbit, as noted in Eq. (3.10), whose nondimensional magnitude is one. Substituting values and differentiating yields the kinematic expansion for the velocity of P_3 in the rotating frame,

$$\dot{\boldsymbol{r}}_{\mathbf{3}} = (\dot{x} - y)\hat{x} + (\dot{y} + x)\hat{y} + \dot{z}\,\hat{z} \tag{3.21}$$

The BKE is again applied for the kinematic expression for the acceleration of P_3,

$$\ddot{\boldsymbol{r}}_{\mathbf{3}} = (\ddot{x} - 2\dot{y} - x)\hat{x} + (\ddot{y} + 2\dot{x} - y)\hat{y} + \ddot{z}\hat{z} \tag{3.22}$$

Finally, the vector components of $\ddot{\boldsymbol{r}}_{\mathbf{3}}$ in Eq. (3.22) and Eqs. (3.15)-(3.17) are combined to form the three nonlinear nondimensional second-order differential equations of motion for P_3,

$$\ddot{x} - 2\dot{y} - x = -\frac{(1-\mu)(x+\mu)}{r_{13}^3} - \frac{\mu(x-1+\mu)}{r_{23}^3} \tag{3.23}$$

$$\ddot{y} + 2\dot{x} - y = -\frac{(1-\mu)y}{r_{13}^3} - \frac{\mu\, y}{r_{23}^3} \tag{3.24}$$

$$\ddot{z} = -\frac{(1-\mu)z}{r_{13}^3} - \frac{\mu\, z}{r_{23}^3} \tag{3.25}$$

There is no known analytical solution to these equations of motion. Thus, numerical integration methods are leveraged to produce the time history for a particle originating from an initial state.

3.2.4 Equations of Motion Derivation from Hamilton's Equations

The equations of motion for the CR3BP are also derived from Hamiltonian dynamics. Consider the generalized coordinates x, y, z. The scalar specific kinetic energy is computed as,

$$T = \frac{1}{2}\left[(\dot{x}-y)^2 + (\dot{y}+x)^2 + \dot{z}^2\right] \tag{3.26}$$

Next, the specific gravitational potential energy for P_3 is defined as,

$$U = -\left(\frac{1-\mu}{r_{13}} + \frac{\mu}{r_{23}}\right) \tag{3.27}$$

Together, the difference between kinetic and potential energy form the Lagrangian $\mathcal{L} = T - U$,

$$\mathcal{L} = \frac{1}{2}\left[(\dot{x}-y)^2 + (\dot{y}+x)^2 + \dot{z}^2\right] + \frac{1-\mu}{r_{13}} + \frac{\mu}{r_{23}} \tag{3.28}$$

The generalized momenta, p_i, associated with the Lagrangian are formed from the derivatives of $\mathcal{L}$ with respect to the generalized coordinates x, y, z, i.e.,

$$p_x = \dot{x} - y \qquad p_y = \dot{y} + x \qquad p_z = \dot{z} \tag{3.29}$$

The Hamiltonian is formulated as $\mathcal{H} = p_x\dot{x} + p_y\dot{y} + p_z\dot{z} - \mathcal{L}$, that simplifies to,

$$\mathcal{H} = \frac{1}{2}v^2 - \frac{1}{2}\left(x^2 + y^2\right) - \frac{1-\mu}{r_{13}} - \frac{\mu}{r_{23}} \tag{3.30}$$

where $v^2 = \dot{x}^2 + \dot{y}^2 + \dot{z}^2$ is the velocity magnitude squared. The expression for $\mathcal{H}$ is rewritten as a function of the generalized coordinates and generalized momenta using Equation (3.29),

$$\mathcal{H} = \frac{1}{2}\left((p_x + y)^2 + (p_y - x)^2 + p_z^2\right)^2 - \frac{1}{2}\left(x^2 + y^2\right) - \frac{1-\mu}{r_{13}} - \frac{\mu}{r_{23}} \tag{3.31}$$

Hamilton's canonical equations are applied to solve for the equations of motion that govern P_3. Hamilton's equations are defined as,

$$\frac{dp_i}{dt} = -\frac{\partial \mathcal{H}}{\partial q_i} \tag{3.32}$$

where p_i are the generalized momenta, and q_i are the generalized coordinates. Differentiating Eq. (3.29) with respect to time, and substituting variables, yields the relationships,

$$\ddot{x} = -\frac{\partial \mathcal{H}}{\partial x} + \dot{y} \qquad \ddot{y} = -\frac{\partial \mathcal{H}}{\partial y} - \dot{x} \qquad \ddot{z} = -\frac{\partial \mathcal{H}}{\partial z} \tag{3.33}$$

The partial derivatives of $\mathcal{H}$ are computed by differentiating Eq. (3.31), and combined with Eq. (3.29). After simplification, the partial derivatives of the Hamiltonian with respect to generalized coordinates are,

$$\frac{\partial \mathcal{H}}{\partial x} = -\dot{y} - x + \frac{(1-\mu)(x+\mu)}{r_{13}^3} + \frac{\mu(x+1-\mu)}{r_{23}^3} \tag{3.34}$$

$$\frac{\partial \mathcal{H}}{\partial y} = \dot{x} - y + \frac{(1-\mu)y}{r_{13}^3} + \frac{\mu\, y}{r_{23}^3} \tag{3.35}$$

$$\frac{\partial \mathcal{H}}{\partial z} = \frac{(1-\mu)z}{r_{13}^3} + \frac{\mu\, z}{r_{23}^3} \tag{3.36}$$

With the partials of the Hamiltonian computed, Eqs. (3.34)-(3.36) are substituted into Eq. (3.33),

$$\ddot{x} - 2\dot{y} - x = -\frac{(1-\mu)(x+\mu)}{r_{13}^3} - \frac{\mu(x-1+\mu)}{r_{23}^3} \tag{3.37}$$

$$\ddot{y} + 2\dot{x} - y = -\frac{(1-\mu)y}{r_{13}^3} - \frac{\mu\, y}{r_{23}^3} \tag{3.38}$$

$$\ddot{z} = -\frac{(1-\mu)z}{r_{13}^3} - \frac{\mu\, z}{r_{23}^3} \tag{3.39}$$

As expected, this result is identical to the equations of motion derived directly from Newton's second law, listed in Eqs. (3.23)-(3.25).

3.2.5 Integral of Motion

To analytically integrate the CR3BP equations of motion, 12 integration constants are required. With only 10 known constants, no closed-form analytical solution exists. While a full analytical solution is not available, one useful integral of motion emerges in the CR3BP formulation. This scalar term, denoted the **Jacobi constant**, C, offers useful insight into the orbital energy as formulated in the CR3BP rotating frame. This quantity is constructed by noting that the system is conservative and, hence, the force acting on P_3 is the gradient of a potential function, that is, in the inertial frame,

$$m_3\ddot{\boldsymbol{r}}_{\boldsymbol{3}} = \nabla \boldsymbol{U} \tag{3.40}$$

where U is the gravitational potential function defined in Eq. (3.27). The CR3BP equations of motion are derived in a rotating reference frame, so additional terms are included in Eq. (3.40) to accommodate the centrifugal potential. Combining yields the *pseudo-potential* function,

$$U^* = \underbrace{\frac{1-\mu}{r_{13}} + \frac{\mu}{r_{23}}}_{\text{Gravitational}} + \underbrace{\frac{1}{2}(x^2+y^2)}_{\text{Centrifugal}} \tag{3.41}$$

For convenience, U_i^* denotes the first partial derivative $\frac{\partial U^*}{\partial i}$. Similarly, the second partial derivative is denoted $U_{ij}^* = \frac{\partial U_i^*}{\partial j}$. These partials are easily evaluated for each position vector measure number x, y, z,

$$U_x^* = -\frac{(1-\mu)(x+\mu)}{r_{13}^3} - \frac{\mu(x-1+\mu)}{r_{23}^3} + x \tag{3.42}$$

$$U_y^* = -\frac{(1-\mu)y}{r_{13}^3} - \frac{\mu y}{r_{23}^3} + y \tag{3.43}$$

$$U_z^* = -\frac{(1-\mu)z}{r_{13}^3} - \frac{\mu z}{r_{23}^3} \tag{3.44}$$

The pseudo-potential U^* allows the equations of motion to be conveniently written in terms of the terms of U_i^*,

$$\ddot{x} - 2\dot{y} = U_x^* \tag{3.45}$$

$$\ddot{y} + 2\dot{x} = U_y^* \tag{3.46}$$

$$\ddot{z} = U_z^* \tag{3.47}$$

This succinct form of the CR3BP equations of motion offers insight into the motion as influenced by the gravitational potential.

Jacobi Constant Derivation

Multiple methodologies exist for deriving the Jacobi constant. Commonly, the Jacobi constant is constructed from the dot product between the velocity of P_3 in the rotating frame and the vector gradient of U^*,

$$\dot{\boldsymbol{r}}_{\boldsymbol{3}} \cdot \nabla \boldsymbol{U^*} = U_x^*\dot{x} + U_y^*\dot{y} + U_z^*\dot{z} = \dot{x}(\ddot{x} - 2\dot{y}) + \dot{y}(\ddot{y} + 2\dot{x}) + \dot{z}\,\ddot{z} \tag{3.48}$$

The pseudo-potential U^* is a function of position only and, thus, the left side of Eq. (3.48) simplifies to the scalar time derivative of U^*, resulting in the relationship,

$$\frac{dU^*}{dt} = \dot{x}(\ddot{x} - 2\dot{y}) + \dot{y}(\ddot{y} + 2\dot{x}) + \dot{z}\,\ddot{z} \tag{3.49}$$

The relationship in Eq. (3.49) is then integrated directly, yielding,

$$\frac{1}{2}(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) = U^* + \tilde{C} \tag{3.50}$$

where $\tilde{C}$ is the constant of integration. By convention, the Jacobi constant, C, is defined as positive such that $C = -2\tilde{C}$,

$$C = 2U^* - v^2 \tag{3.51}$$

The existence of an integration constant allows for additional insight into the dynamical response in the CR3BP.

Relationship Between Jacobi Constant and the Hamiltonian

The Jacobi constant is derived by directly integrating the expression in Eq. (3.48), however, other approaches exist for deriving C. In particular, the equations of motion are also straightforwardly derived from the Hamiltonian, Eq. (3.30), which results in the equations of motion in Eqs. (3.37)-(3.39). One advantage of a Hamiltonian approach is the immediate emergence of the integration constant by observing that $\mathcal{H}$ is a function of the generalized coordinates and generalized momenta, but is not an explicit function of time. Therefore, $\mathcal{H}$ remains constant through integration, and is evaluated as,

$$\mathcal{H} = \frac{1}{2}v^2 - \frac{1}{2}\left(x^2 + y^2\right) - \frac{1-\mu}{r_{13}} - \frac{\mu}{r_{23}} \tag{3.52}$$

The expression for $\mathcal{H}$ is expressed more concisely by writing it in terms of U^*,

$$\mathcal{H} = \frac{1}{2}v^2 - U^* \tag{3.53}$$

The relationship between $\mathcal{H}$ and C is clear by comparing Eqs. (3.53) and (3.51), that is, $C = -2\mathcal{H}$.

3.2.6 Equilibrium Solutions

In the CR3BP, equilibrium solutions are frequently labelled the *Lagrange points* or *libration points*, and are determined when the energy gradient is equal to zero, $\nabla \boldsymbol{U^*} = \mathbf{0}$. The scalar components of the energy gradient are,

$$U_x^* = -\frac{(1-\mu)(x+\mu)}{r_{13}^3} - \frac{\mu(x-1+\mu)}{r_{23}^3} + x = 0 \tag{3.54}$$

$$U_y^* = y\left(1 - \frac{1-\mu}{r_{13}^3} - \frac{\mu}{r_{23}^3}\right) = 0 \tag{3.55}$$

$$U_z^* = -z \underbrace{\left[\frac{1-\mu}{r_{13}^3} + \frac{\mu}{r_{23}^3}\right]}_{>0} = 0 \tag{3.56}$$

The second term in Eq. (3.56) is positive, i.e., $\frac{1-\mu}{r_{13}^3} + \frac{\mu}{r_{23}^3} > 0$. Therefore, the equation is satisfied only if $z = 0$. Hence, all equilibrium points are located within the $\hat{x}$-$\hat{y}$ plane. The two conditions that satisfy Eq. (3.55) are,

$$y = 0 \qquad \text{or} \qquad 1 - \frac{1-\mu}{r_{13}^3} - \frac{\mu}{r_{23}^3} = 0 \tag{3.57}$$

These cases are solved separately, with $y = 0$ corresponding to the *collinear* libration points, and $1 - \frac{1-\mu}{r_{13}^3} - \frac{\mu}{r_{23}^3} = 0$ producing the *triangular* libration points.

Collinear Libration Points

In the CR3BP, three collinear equilibrium solutions exist along the $\hat{x}$ axis; these arise from Eq. (3.55) for the case $y = 0$. The collinear libration points are determined by rearranging Eq. (3.54), and solving the transcendental equation,

$$x = \frac{(1-\mu)(x+\mu)}{r_{13}^3} + \frac{\mu(x-1+\mu)}{r_{23}^3} \tag{3.58}$$

From among various options, Eq. (3.58) is solved here numerically using a Newton-Raphson method. To distinguish between points, it is useful to iteratively solve for a distance with respect to a primary body, that is, rather than solving for x explicitly, each libration point is located by iteratively solving for the value of γ_i, as illustrated in Figure 3.3, where L_i denotes the libration point with index i. This substitution allows the direct evaluation of $x_{L_1} = (1-\mu-\gamma_1)$, $x_{L_2} = 1-\mu+\gamma_2$, and $x_{L_3} = -\mu-\gamma_3$.

Using the definition of γ_i, Equation (3.58) is reformulated in terms of γ_i. For each collinear libration point, these are,

$$1-\mu-\gamma_1 = \frac{(1-\mu)}{(1-\gamma_1)^2} - \frac{\mu}{\gamma_1^2} \tag{3.59}$$

$$1-\mu+\gamma_2 = \frac{1-\mu}{(1+\gamma_2)^2} + \frac{\mu}{\gamma_2^2} \tag{3.60}$$

$$\mu+\gamma_3 = \frac{(1-\mu)}{\gamma_3^2} + \frac{\mu}{(1+\gamma_3)^2} \tag{3.61}$$

These equations are solved using a basic Newton-Raphson method to produce three collinear libration points. By convention, L_1 is located between the primaries, L_2 is on the $+\hat{x}$ side of P_2, and L_3 is the remaining point in the $-\hat{x}$ direction from P_1, as depicted in Figure 3.3.

Triangular Libration Points

In the CR3BP, two equilibrium solutions are computed analytically. These solutions are termed the triangular libration points due to their geometry, and are labelled

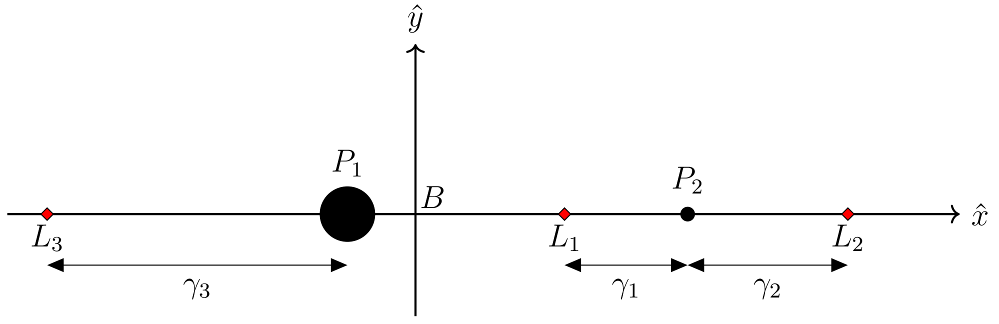


Figure 3.3. Collinear Lagrange points for three-body system with $\mu = 0.2$.

L_4 and L_5, as illustrated in Figure 3.4. The triangular libration points are derived from Eq. (3.55), in the case where,

$$1 - \frac{1-\mu}{r_{13}^3} - \frac{\mu}{r_{23}^3} = 0 \tag{3.62}$$

Since r_{13} and r_{23} are defined as physical quantities, this equation is only satisfied when the values for r_{13} and r_{23} possess no imaginary component; such a result only occurs for $r_{13} = r_{23} = 1$. Substituting $r_{13} = r_{23} = 1$ into Eqs. (3.18) and (3.19) (given $z = 0$) produces,

$$1 = (x_{L_t} + \mu)^2 + y_{L_t}^2 \tag{3.63}$$

$$1 = (x_{L_t} - 1 + \mu)^2 + y_{L_t}^2 \tag{3.64}$$

Subtracting Eq. (3.64) from Eq. (3.64), and simplifying, yields the relationship,

$$x_{L_t} + \mu = \pm(x_{L_t} - 1 + \mu) \tag{3.65}$$

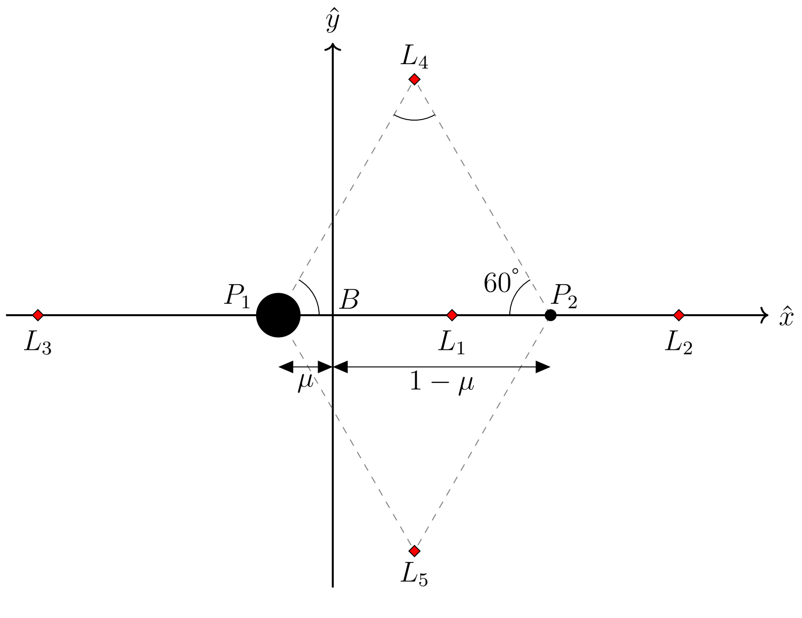


Figure 3.4. Libration points for a three-body system where $\mu = 0.2$.

The positive sign implies a contradiction $(0 = -1)$, clearly an invalid solution. Rather, consider the negative sign such that $x_{L_t} + \mu = -x_{L_t} + 1 - \mu$. This result is then simplified, with the final relationship,

$$x_{L_t} = \frac{1}{2} - \mu \tag{3.66}$$

The value of $x_{L_t}$is then substituted back into Eq. (3.63) and, once simplified, yields the $\hat{y}$-component for the two triangular libration points,

$$y_{L_t} = \pm\frac{\sqrt{3}}{2} \approx 0.866025, \text{ nondim} \tag{3.67}$$

The locations of the triangular libration points, along with the collinear libration points, are illustrated in Figure 3.4.

3.2.7 Zero Velocity Surfaces

A key insight from the Jacobi constant is a set of limiting boundaries for possible motion of the particle P_3. By examining the scenario where relative velocity is equal to zero, $(v = 0)$, for a given value of the Jacobi constant, the Zero Velocity Surface (ZVS) emerges. The ZVS is a 3-D surface that reflects the boundary between the relative velocity values that are real and those that are imaginary for a specific Jacobi constant value. Of course, only real velocity values are physically possible and, hence, the ZVS produces “forbidden” regions of space that are inaccessible to the spacecraft. The ZVS is computed by setting velocity to zero, thus, Eq. (3.51) reduces to,

$$C = 2U^* \tag{3.68}$$

Points that satisfy Eq. (3.68) are determined by noting that U^* is simply a function of position and, thus, all solutions exist in configuration space. With all such locations specified, the ZVS is computed. Frequently, it is simpler to view the ZVS that exists in the $\hat{x}$-$\hat{y}$ plane, called the Zero Velocity Curve (ZVC). The ZVC and ZVS at various energy levels in the Earth-Moon system are displayed in Table 3.2. As energy increases, portals surrounding the Lagrange points open to enable access to different regions in the system space. For example, if P_3 has a Jacobi constant such that $C_{L_1} < C < C_{L_2}$, Table 3.2(c) indicates that P_3 may transit between P_1 and P_2, but is unable to escape the system entirely.

3.3 Circular Restricted Three-Body Problem with Low-Thrust

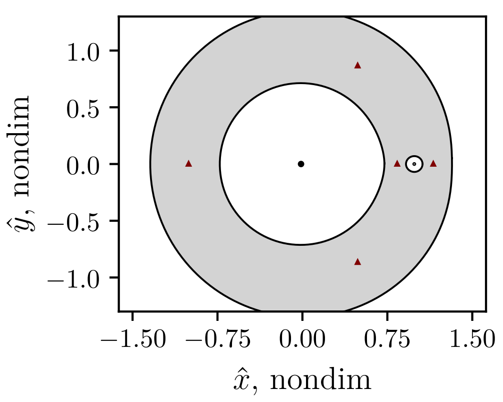
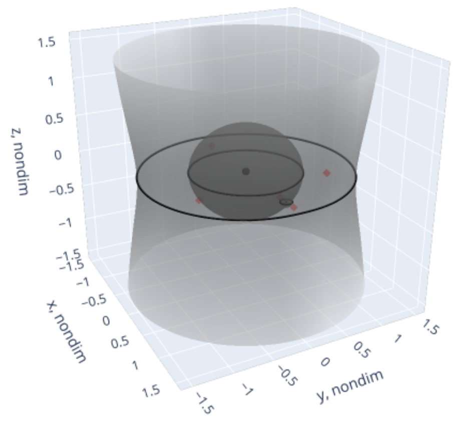
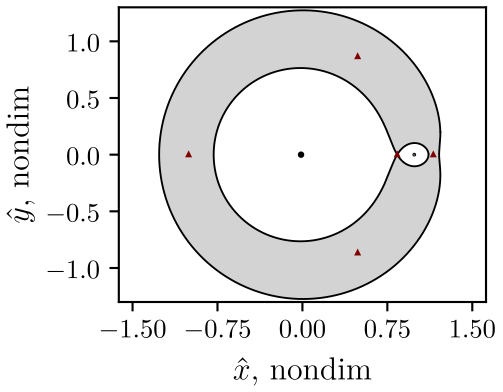
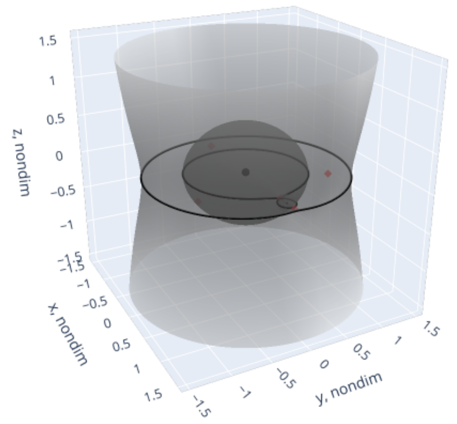
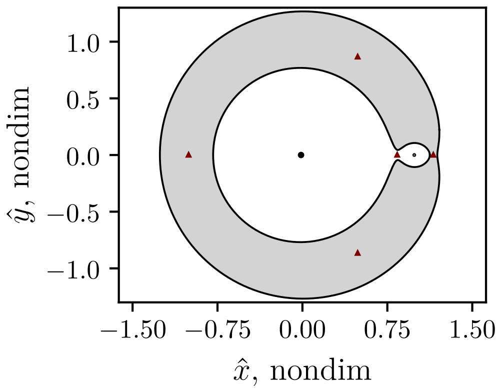
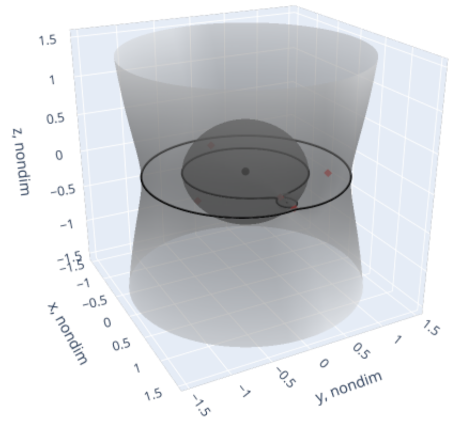
Many mission architectures benefit from the inclusion of low-thrust electric propulsion. In contrast to traditional chemical engines, electric propulsive engines deliver energy changes over much longer time intervals. Low-thrust engines using Solar Electric Propulsion (SEP) include ion thrusters, which are powered through solar panels on the spacecraft. Currently, ion thrusters are successfully employed, for example, on Deep Space 1 [57] and Dawn [58], as well as other missions. Building on this progress, the Power and Propulsion Element (PPE) for the upcoming Gateway mission plans to employ SEP on-board [59].

As detailed by Cox et al. [60], the motion of a low-thrust spacecraft is modeled by augmenting the natural CR3BP equations of motion, defined in Eqs. (3.23)-(3.25). Assuming the low-thrust engine delivers acceleration in kilonewtons, denoted F, the nondimensional low-thrust magnitude, f, is computed as,

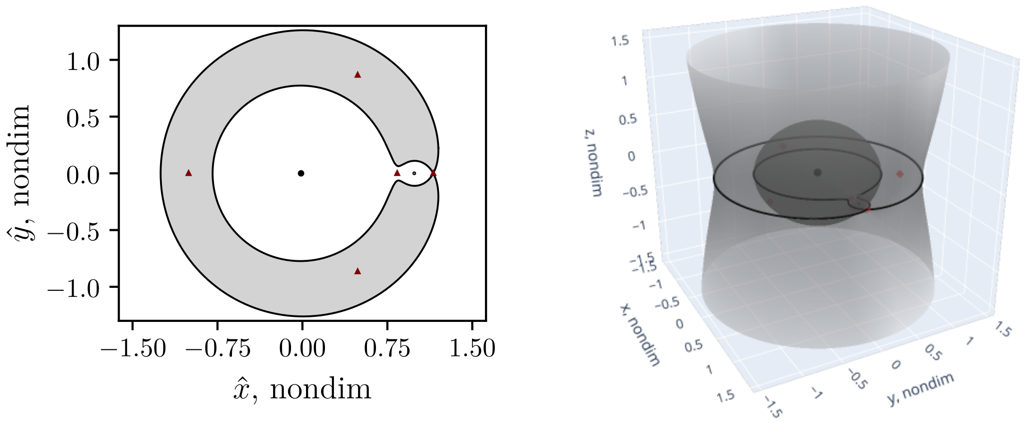
$$f = \frac{F t^{*2}}{l^* M_{3,0}} \tag{3.69}$$

where $M_{3,0}$ is the initial mass of the spacecraft in kilograms. 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. This investigation involves 20.87 hour segments, and the precession of the rotating frame during these time intervals may be addressed when transferring CR3BP solutions into a higher-fidelity model. Furthermore, propulsive capability is inversely related to spacecraft mass and,

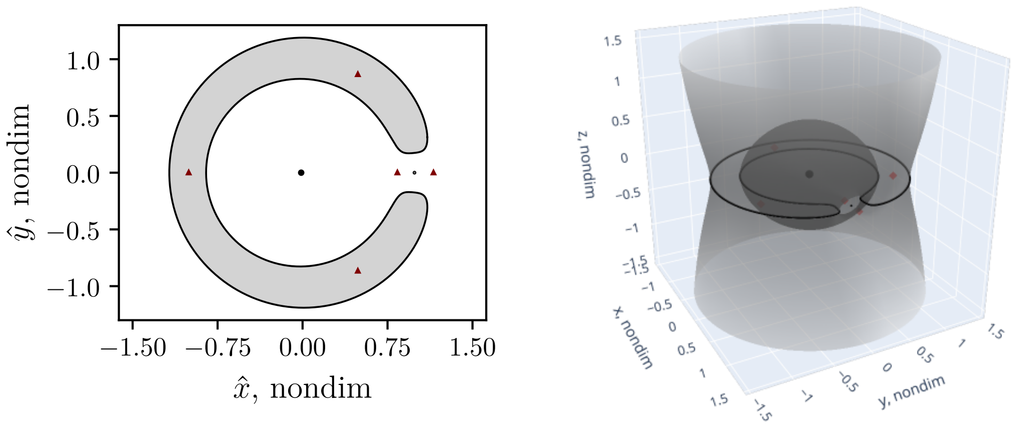
Table 3.2. ZVC and ZVS for various energy levels in the Earth-Moon system.

(a) L_1 portal closed: $C > C_{L_1}$	
	
(b) L_1 portal opening: $C = C_{L_1}$	
	
(c) L_1 portal open: $C_{L_1} < C < C_{L_2}$	
	

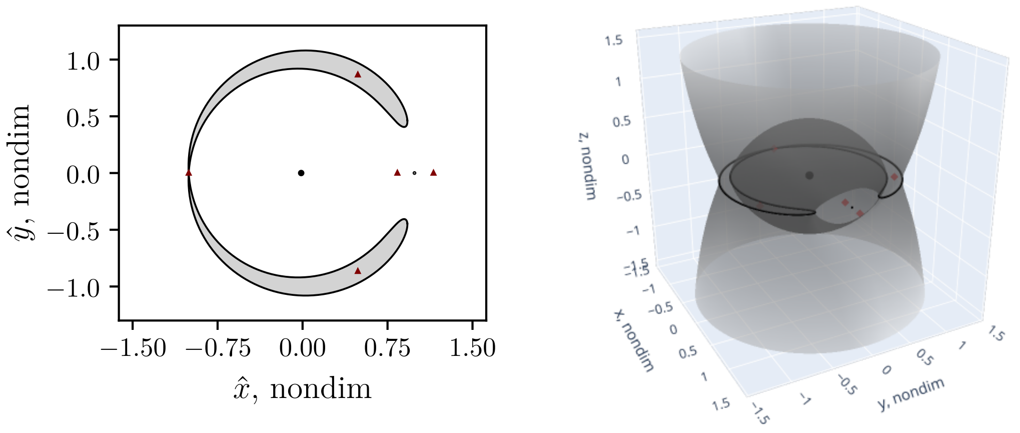
(d) L_2 portal opening: $C = C_{L_2}$



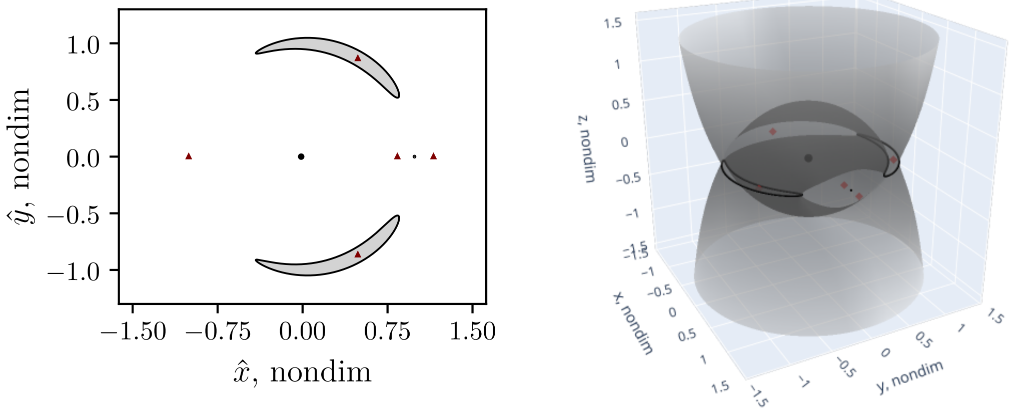
(e) L_1, L_2 portals open: $C_{L_2} < C < C_{L_3}$



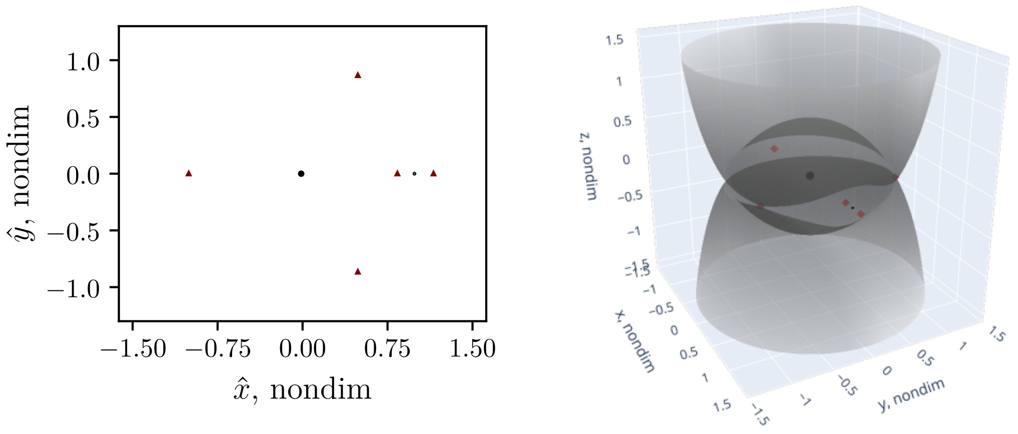
(f) L_3 portal opening: $C = C_{L_3}$



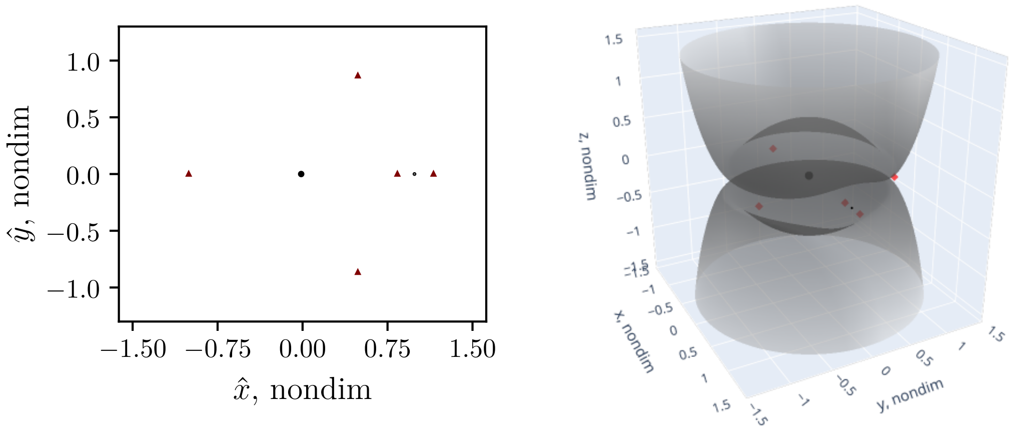
(g) L_1, L_2, L_3 portals open: $C_{L_3} < C < C_{L_4,L_5}$



(h) ZVC disappearing at $C = C_{L_4,L_5}$



(i) No ZVC in the x-y plane: $C < C_{L_4,L_5}$



hence, as propellant is expended, the spacecraft gains more thrust. Therefore, the complete low-thrust acceleration vector is defined as,

$$\boldsymbol{a}_{\text{lt}} = a_{\text{lt}}\hat{u} = \frac{f}{m}\hat{u} = (a_{\text{lt}}u_x)\hat{x} + (a_{\text{lt}}u_y)\hat{y} + (a_{\text{lt}}u_z)\hat{z} \tag{3.70}$$

where $m = M_3/M_{3,0}$ is the nondimensional spacecraft mass, and M_3 is the mass of the spacecraft at the beginning of a thrusting segment. Including variable mass indicates an additional equation of motion is required. Assuming a Constant Specific Impulse (CSI) engine, the rate of change of m with respect to nondimensional time is computed as,

$$\dot{m} = \frac{-fl^*}{I_{\text{sp}}g_0t^*} \tag{3.71}$$

where I_{sp} is the engine's specific impulse (in seconds) and $g_0 = 9.80665 \times 10^{-3}$ km/s is the standard acceleration due to gravity. The motion of P_3 in the low-thrust CR3BP is governed by three second-order differential equations, with an additional first-order differential equation for mass,

$$\ddot{x} - 2\dot{y} - x = -\frac{(1-\mu)(x+\mu)}{r_{13}^3} - \frac{\mu(x-1+\mu)}{r_{23}^3} + a_{\text{lt}}u_x \tag{3.72}$$

$$\ddot{y} + 2\dot{x} - y = -\frac{(1-\mu)y}{r_{13}^3} - \frac{\mu y}{r_{23}^3} + a_{\text{lt}}u_y \tag{3.73}$$

$$\ddot{z} = -\frac{(1-\mu)z}{r_{13}^3} - \frac{\mu z}{r_{23}^3} + a_{\text{lt}}u_z \tag{3.74}$$

$$\dot{m} = 0 + \frac{-fl^*}{I_{\text{sp}}g_0t^*} \tag{3.75}$$

where red signifies the additional terms included for low-thrust force. This investigation includes a sample spacecraft with propulsive capability of $f = 4 \cdot 10^{-2}$. A comparison between this sample spacecraft and other previous and planned engine capabilities is summarized in Table 3.3. The sample spacecraft incorporates more propulsive capability than Hayabusa 1, Hayabusa 2, Dawn and Lunar IceCube (as currently planned), but less than Deep Space 1.

3.3.1 Numerical Integration

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 Eqs. (3.23)-(3.25), as seven coupled first-order equations that satisfy $\dot{\boldsymbol{\alpha}} = \boldsymbol{F}(\boldsymbol{\alpha}, \tau)$,

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ m \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ 2\alpha_5 + \alpha_1 - -\frac{(1-\mu)(\alpha_1+\mu)}{r_{13}^3} - \frac{\mu(\alpha_1-1+\mu)}{r_{23}^3} + \frac{f}{\alpha_7}u_x \\ -2\alpha_4 + \alpha_2 - \frac{(1-\mu)\alpha_2}{r_{13}^3} - \frac{\mu\alpha_2}{r_{23}^3} + \frac{f}{\alpha_7}u_y \\ -\frac{(1-\mu)\alpha_3}{r_{13}^3} - \frac{\mu\alpha_3}{r_{23}^3} + \frac{f}{\alpha_7}u_z \\ \frac{-fl^*}{I_{\text{sp}}g_0 t^*} \end{bmatrix} \tag{3.76}$$

Here, r_{13} and r_{23} are defined as in Equations (3.13) and (3.14), respectively, with $\alpha_1, \alpha_2, \alpha_3$ substituted for x, y, z. These seven equations can be solved using numeric

Abbrv.	Spacecraft	f, nondim	$M_{3,0}$, kg	F, mN
H1	Hayabusa 1 [61]	$1.640 \cdot 10^{-2}$	510	22.8
H2	Hayabusa 2 [62]	$1.628 \cdot 10^{-2}$	608.6	27.0
LIC	Lunar IceCube [63, 64]	$3.276 \cdot 10^{-2}$	14	1.25
Dawn	Dawn [58]	$2.741 \cdot 10^{-2}$	1217.8	91.0
DS1	Deep Space 1 [57]	$6.940 \cdot 10^{-2}$	486.3	92.0
s/c	Sample Spacecraft	$4 \cdot 10^{-2}$	n/a	n/a

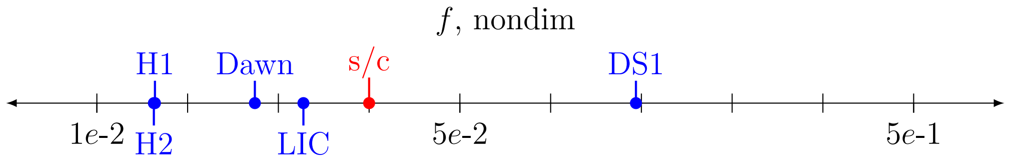


Table 3.3. Low-thrust capability of various spacecraft, nondimensionalized in the Earth-Moon system.

integration techniques. For this investigation, a Runge-Kutta-Fehlberg 78 integration scheme is employed.

4. REINFORCEMENT LEARNING FRAMEWORK

Properly formulating a reinforcement learning environment is critical to algorithmic performance. Researchers develop RL algorithms to be applicable to many types of problems, however each environment must be specifically tailored to a particular application. It is especially important to design an environment such that the underlying assumptions are not violated in the RL process. In particular, the environment quantifies the problem being solved, the system dynamics, the many details of the episodic process, and the process to pass information back-and-forth between the agent and environment. In particular, as depicted in Figure 2.3, the state, action, and reward signals quantify the communications process. Implementing an effective environment involves properly defining each signal. The environment must define the initialization and termination of episodes and must ensure that its computational footprint does not inhibit learning performance.

4.1 Signal Definition

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

4.1.1 State Signal

Under a Markov Decision Process (MDP), the environmental state at time t ($\boldsymbol{s_t}$) must include all necessary past environmental information that impacts the future [3]. For the CR3BP, position, velocity, and spacecraft mass are together sufficient, since future states are predicted by numerically integrating the equations of motion specified in Eqs. (3.72) - (3.75). Hence, at every time step t, the dynamical state $\boldsymbol{q_t}$ is defined as,

$$\boldsymbol{q_t} = \begin{bmatrix} \boldsymbol{\rho}^{\text{agent}} & m \end{bmatrix} = \begin{bmatrix} x & y & \dot{x} & \dot{y} & m \end{bmatrix} \tag{4.1}$$

While $\boldsymbol{q_t}$ alone is sufficient to satisfy the Markov property in this planar problem, the agent performance is greatly enhanced by augmenting the dynamical state, $\boldsymbol{q_t}$, with additional variables to form the state signal, $\boldsymbol{s_t}$. In the PPO formulation, the actor and critic networks receive the complete state signal as inputs, as depicted in Figure 2.4. Hence, both the policy and value functions are dependent on the selection of additional variables. Since this problem involves an agent learning to track a reference solution, relative position and velocity are essential to the agent performance and the ability to extrapolate to nearby motion. Including relative state information is similarly useful for RL in a stationkeeping environment [65]. The relative information is computed simply as,

$$\boldsymbol{\delta\rho} = \boldsymbol{\rho}^{\text{agent}} - \boldsymbol{\rho}^{\text{ref}} = \begin{bmatrix} \delta x & \delta y & \delta\dot{x} & \delta\dot{y} \end{bmatrix} \tag{4.2}$$

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

$$\boldsymbol{\rho}^{\text{ref}} \in \mathcal{R}^{\text{ref}} \quad \text{s.t.} \quad k = |\boldsymbol{\delta\rho}| = \sqrt{\delta x^2 + \delta y^2 + \delta \dot{x}^2 + \delta \dot{y}^2} \quad \text{is minimal} \tag{4.3}$$

The scalar value k is a function of both the position and velocity deviation, as depicted in Figure 4.1. This relative state information, along with the dynamical state and other optional additional observations, form the complete state signal,

$$\boldsymbol{s_t} = \begin{bmatrix} \boldsymbol{q_t} & \boldsymbol{\delta\rho} & \text{additional observations} \end{bmatrix} \tag{4.4}$$

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

The additional observations are problem-dependent and are, thus, included here as optional parameters. Since this investigation involves the CR3BP dynamical model, including some dynamical information in the reward signal is advantageous for learning performance. In particular, the Jacobi constant, defined in Eq. (3.51), communicates energy deviations to the actor and critic networks. At each time step, the Jacobi constant for $\boldsymbol{s_t}$, denoted C_{s_t}, is computed for a particular state, and then combined with the Jacobi constant value from the reference trajectory, C_{ref}, to form

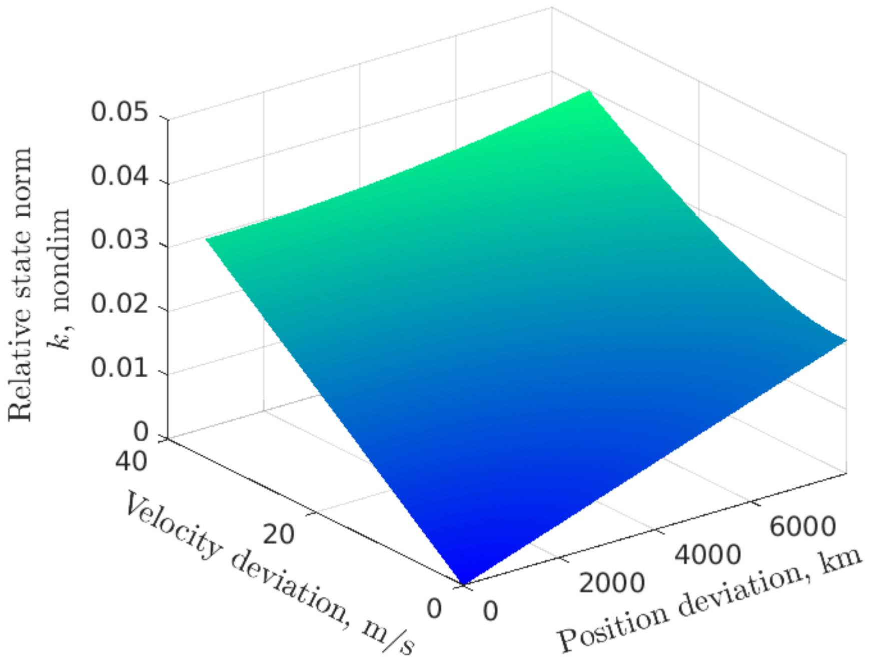


Figure 4.1. Relative position and velocity norm values, k. Position and velocity values are nondimensionalized in the Earth-Moon system, and k is computed as the L2 norm $k = \sqrt{\delta r^2 + \delta v^2}$.

the additional observations in Eq. (4.4). The complete state vector in the CR3BP environment is then defined as,

$$\boldsymbol{s_t} = \begin{bmatrix} \boldsymbol{q_t} & \boldsymbol{\delta\rho} & C_{s_t} & C_{\text{ref}} \end{bmatrix} \tag{4.5}$$

Omitting the Jacobi constant from the state signal entirely does not prohibit convergence, but the resulting policy is less optimal than one that incorporates the constant. The beneficial impact of including the Jacobi constant indicates that, when applying this approach to other dynamical models where the Jacobi integral may not be available, additional energy-like observations may prove advantageous.

4.1.2 Action Signal

In any MDP, an agent influences the environment by means of actions. For a low-thrust spacecraft, the action takes the form of a thrust magnitude and direction. While this action does not instantaneously alter the environmental state, its impact is realized in the numerical propagation that occurs between time steps. In this case, the action is identified by the actor network, which outputs both a thrust magnitude, $\tilde{f}$, and the vector components representing thrust direction, $(\tilde{u}_x,\ \tilde{u}_y)$, as depicted in the output layer of the actor network in Figure 2.4. During the training phase, the network outputs the mean value of each action parameter and uses these in conjunction with a derived variance to create a normal distribution for each value. The mean is essentially the agent's best guess for the action given a particular observation, and the variance is included to encourage exploration. Miller et al. employ a similar action definition for interplanetary trajectory generation [35]. As in all policy optimization RL methods, over the course of training, the output of the network approaches an optimal policy. Once fully trained, exploration is no longer necessary, so the mean values are used directly to form a deterministic controller.

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

$$f = \frac{\tilde{f} + 1}{2} f_{\max} \in [0, f_{\max}] \tag{4.6}$$

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

$$\boldsymbol{a_t} = \begin{bmatrix} f & u_x & u_y \end{bmatrix} \quad \text{such that} \quad \hat{u} = [u_x \;\; u_y] = \frac{[\tilde{u}_x \;\; \tilde{u}_y]}{\sqrt{\tilde{u}_x^2 + \tilde{u}_y^2}} \tag{4.7}$$

While parameterizing thrust as a unit vector/magnitude is straightforward, a potential drawback is an equation of constraint that is unknown to the controller, i.e., thrust direction is normalized after neural network evaluation. While it seems appealing to reformulate the low-thrust parameterization to eliminate this constraint, note that including an angle as an output value for any bounded activation function results in a critical discontinuity in the available action space and, therefore, in the gradient of the action with respect to the observations. This discontinuity occurs because, once re-scaled to a range $[0, 2\pi]$, the agent cannot perform an update step to push the output angle past either end bound. Hence, while parameterizations that include angles are potentially beneficial for other applications, such as trajectory design [60] and targeting [66], the bounded action implies that an alternate approach is required for this application.

An alternative low-thrust parameterization that has been applied to PPO by Miller and Linares is empowering the agent to command the thrust in each direction independently, such that each direction is allowed to employ the maximum allowable thrust [34]. While this approach avoids the discontinuity issue associated with angles, the drawback is that a physical engine has an associated total maximum thrust, but does not possess a practical limitation on the thrust direction of individual vector components. For the action to be more reflective of a physical engine, the magnitude/unit vector formulation, detailed here, separates thrust magnitude and direction in the action output. While the external equation of constraint that accompanies this strategy causes repetition in possible actions, it does not prohibit convergence to an effective policy.

4.1.3 Reward Signal

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

$$r = \begin{cases} \eta e^{-\lambda k} & \sqrt{\delta x^2 + \delta y^2} < \delta r_{\max} \text{ and } \sqrt{\delta \dot{x}^2 + \delta \dot{y}^2} < \delta v_{\max} \\ p & \text{deviate from reference or impact} \end{cases} \tag{4.8}$$

where λ is a scaling factor that increases the gradient of the reward, k is defined in Eq. (4.3) as the relative distance (in position and velocity) to the nearest point along the reference, and p is a penalty for deviating from the reference or impacting the primary or secondary body. Finally, η is a scaling term evaluated as,

$$\eta = \frac{i}{n}\xi + 1 \tag{4.9}$$

where i is the index of the reference trajectory state, n is the size of the set of states along the reference trajectory, $\mathcal{R}^{\text{ref}}$, and ξ is a tuning variable to adjust the rate at which the reward increases along the reference. If the nearest neighbor is along the arrival orbit and not the reference trajectory, then η is assumed to be a maximum value $\eta_{\text{arrival}} = \eta_{\max} = \xi + 1$. Formulating the reward signal to be at maximum when the agent reaches its target encourages the agent to fully complete the given transfer. The reward as a function of k is plotted in Figure 4.2. As depicted in Figure 4.1, for error thresholds of $\delta r_{\max} = 8000$ km and $\delta v_{\max} = 35$ m/s, η yields a maximum value $\eta \approx 0.04$, which implies that $\eta \in [0, 0.04]$. A value of $\eta > 0.04$ indicates a penalty is always applied, however, the penalty may still be applied in the case that position or velocity has individually violated a threshold. Figure 4.2 illustrates the change in the reward magnitude based on both the relative distance to the reference,

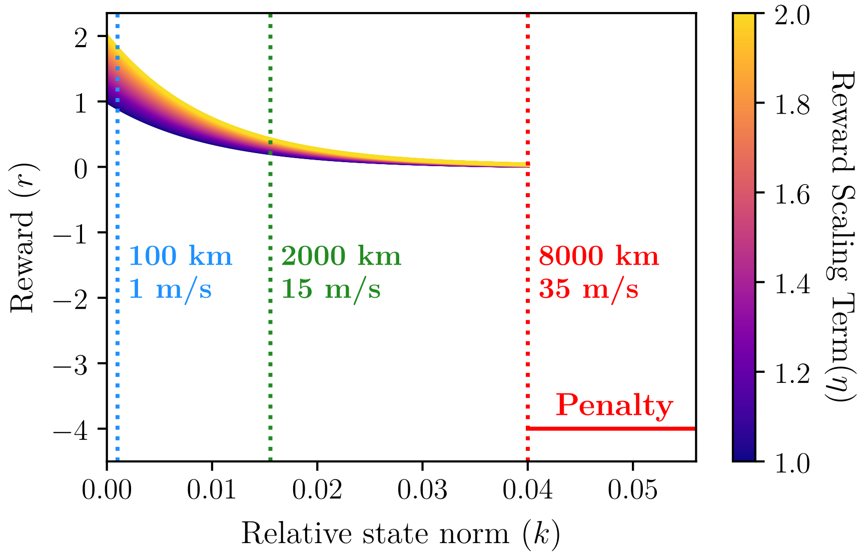


Figure 4.2. Reward function for $\lambda = 100$ plotted over k with varying η, where $\eta = 1$ indicates the beginning of the reference trajectory and $\eta = 2$ indicates the arrival orbit. A penalty of $p = -4$ is imposed when an error threshold is met for position or velocity (8000 km and 35 m/s, respectively). The vertical red line indicates the maximum value of k before the error is imposed, though it may be imposed earlier if either position or velocity violates the threshold.

and the progress along the transfer. For example, if the nearest neighbor along the reference trajectory possesses the values 100 km, 1 m/s, then the reward varies from approximately 0.9 to 1.9 based on the location of the nearest neighbor. The range of possible values narrows as the relative distance decreases, until a penalty is imposed.

The reward function employed in this investigation differs from that in Miller and Linares [34] by removing time from the equation. In their formulation, a spacecraft is rewarded for arriving in a periodic orbit at a specific time. While requiring a matching time is important for many applications, such as rendezvous, other scenarios do not warrant this constraint. With a time-autonomous reward function, the agent returns to a reference trajectory, without penalty for a slightly longer or shorter transfer time. The reward function also differs slightly from LaFarge et al. [43] by removing the 'bonus' for reaching an arrival criteria and by combining the penalties for deviation and planetary impact into one single scalar value.

The measurement of nearness in the reward function, as detailed in Eq. (4.8), is visualized in Figure 4.3. To illustrate the region of high reward surrounding the reference, perturbations are introduced in y_0 at the point where the trajectory crosses the $x = 1 - \mu$ plane. As each of these perturbed states is propagated forward in time, their deviation off the reference is visualized by the reward colormap. Once deviation beyond the threshold occurs, the trajectory is colored light gray to denote areas where a penalty is imposed. Due the exponential term in Eq. (4.8), high reward exists solely in the region immediately surrounding the reference. Hence, to continue accruing reward, the agent is encouraged to maintain close proximity to the reference path.

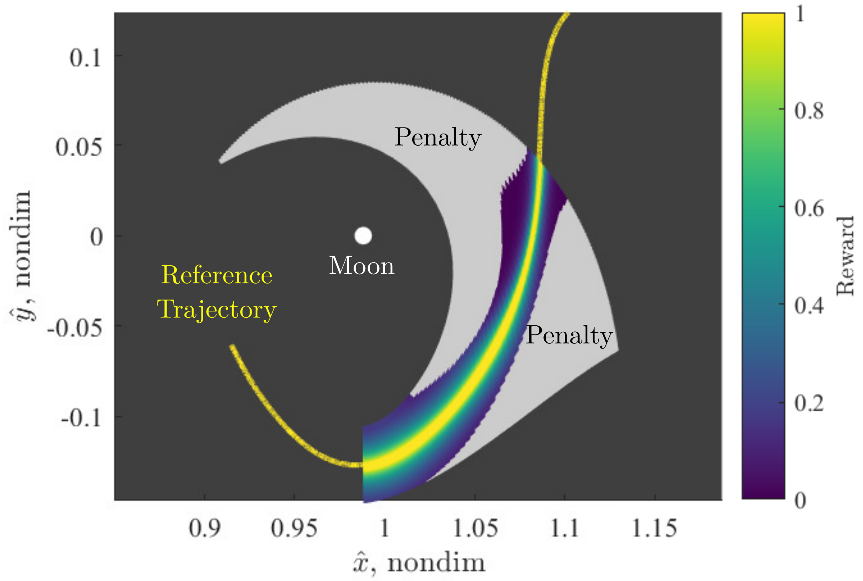


Figure 4.3. Motion nearby a reference trajectory originating at the plane defined by $x = 1 - \mu$. Perturbations are introduced in y_0, and then propagated without thrust. Each state is colored based on the reward function defined by Eq. (4.8), where $\lambda = 100$, $\eta = 1$, and the maximum deviations in position and velocity are 8000 km and 35 m/s, respectively (from LaFarge et al., Ref [43], Fig. 3).

A notable limitation of this time-autonomous approach is a reference trajectory that includes a departure periodic orbit. In this case, the reward function, as defined here, causes an agent to learn to maximize future reward by stationkeeping about the departure orbit rather than proceeding along the reference. To combat this behavior, the variable η is leveraged in Eq. (4.8) to encourage the agent to continue along the reference, and discourage the initial periodic behavior. While η discourages initial stationkeeping behavior, it does not entirely eliminate the tendency. Like other optimization methods, abundant local minima encourage an agent to converge to sub-optimal behavior. Including η discourages this behavior, but does not entirely eliminate the possibility of sub-optimal convergence and return to the stationkeeping local minimum.

4.2 Description of an Episode

An episode originates by selecting a random initial state along the departure periodic orbit from a uniform distribution and introducing a perturbation in position and velocity. The nondimensional mass for the initial state is assumed to be one. The perturbation is sampled from two normal distributions (position and velocity), where each component of $\boldsymbol{\rho}_0$ is perturbed individually with standard deviations in position and velocity of σ_r and σ_v, respectively. The averages across the perturbation distributions are all zero, and σ is varied depending on desired disturbance for a particular simulation.

Once an initial state is generated, the departure orbit is no longer employed in the simulation, and the agent is required to follow its reference trajectory to accrue reward. Given the perturbed initial state, the agent computes an action, i.e., a thrust direction and magnitude. The equations of motion are then propagated within the environment for a particular time horizon (Δt). The specified Δt between the actions is an important input selection for the agent performance. If Δt is too large, then the nonlinearities become more pronounced, and the agent is offered fewer opportunities for sufficient actions over an episode. Furthermore, thrust direction is fixed in the rotating frame between segments and, thus, longer time horizons present additional difficulty in transferring the control solution to an inertial-fixed scheme. However, if Δt is too small, there is not time for the thrust direction to demonstrate a discernible impact on the system. For the examples included in this investigation, $t = 0.2$ nondim ≈ 20.87 hrs strikes a balance between the extrema of being too large or too small. After propagating over Δt again, the agent again selects a new action. Actions are introduced sequentially, after each time interval, until the agent deviates from the reference, impacts a planetary body, arrives at the target orbit, or reaches a maximum number of time steps.

4.3 Nearest Neighbor Searching

The computation of the relative state for the reward and state signals presents some practical challenges in implementation. First, the nearest reference state must be selected from a discrete set of states along the reference path, $\mathcal{R}^{\text{ref}}$. For an accurate assessment of nearness, the trajectory must include a large number of states. However, since the reward is computed at every time step, a brute force search through $\mathcal{R}^{\text{ref}}$ is computationally infeasible. To ease this computational burden, the nearest neighbor search is instead conducted by traversing through a K-dimensional tree (KD-Tree). A KD-Tree is a data structure frequently used for data clustering in unsupervised learning applications. This specialized type of binary search tree reduces the algorithmic complexity of the nearest neighbor problem from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$, a significant improvement when n is large. In this investigation, Scikit-learn's `neighbors.KDTree` implementation is employed to facilitate the nearest neighbor search process [67]. A similar approach is successful for autonomously locating neighbors in higher dimensional Poincaré maps [20, 21]. This application differs in that the neighbor search is conducted for only a single state, rather than intersections from two discrete sets.

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

5. MISSION APPLICATION: LIBRATION POINT TRANSFERS

To illustrate the performance for the PPO-generated controller, several sample scenarios are considered. Due to the recent increased interest in cislunar space, the Earth-Moon CR3BP system serves as the basis for the sample cases. The specific characteristic quantities for this system are listed in Table 5.1. The low-thrust propulsion model assumes a Constant Specific Impulse (CSI) engine with parameters listed in Table 5.2. A comparison between this sample engine and various existing spacecraft is detailed in Table 3.3. The sample spacecraft possesses more propulsive capability than Hayabusa, Dawn, and Lunar IceCube, but less than Deep Space 1 [60]. Hence, the sample spacecraft is consistent with current low-thrust capabilities.

For the sample scenarios, planar transfers between orbits in the vicinity of the L_1 and L_2 libration points, with various geometries, serve as illustrative test problems for the proposed guidance framework. In particular, Lyapunov orbits at the same value of Jacobi constant are examined. With energy constrained, motion in the lunar vicinity is bounded in configuration space by a forbidden region, denoted the Zero Velocity Curves (ZVCs) [68]. Jacobi constant levels included in this investigation correspond to forbidden regions similar to those depicted in Table 3.2 (e), where the L_1 and L_2 portals are open, but the L_3 portal is closed. Furthermore, the shared Jacobi constant value between the orbits indicates that heteroclinic transfers may be available. Heteroclinic transfers occur when manifold intersections create continuous Δv-free paths between two periodic orbits. These continuous trajectories are constructed by

Table 5.1. Characteristic quantities in the Earth-Moon System

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

Table 5.2. Low-thrust engine characteristics

$f_{\max}$, nondim	I_{sp}, s
$4 \cdot 10^{-2}$	3000

selecting an initial guess from manifold intersections on a Poincaré map and corrected using the methodology detailed by Haapala and Howell [4]. Heteroclinic transfers are one of the few cases in the CR3BP where globally optimal geometries exist and, thus, provide a useful test framework for the controller since no maneuvers are required.

While the agent is trained using only one reference trajectory, the controller's ability to generalize control histories to other geometries in the lunar region is investigated. In reality, a variety of factors cause a planned path to shift in-flight. For example, on-board targeting yields trajectory corrections and a nearby solution is generated. However, in generating nearby transfers, it is often difficult to produce any initial guess for the control history. In particular, for Orion, Trajectory Correction Maneuvers (TCMs) are nominally zero [7]. However, as perturbations cause a spacecraft to deviate and these maneuvers become necessary, zero thrust is a poor initial guess, and likely negatively impacts the performance of the targeter. To address this limitation, despite no training with other reference geometries, the ability for the proposed controller to generalize past experience is demonstrated. If a controller is only applicable to the particular reference it has seen, and the training process requires significant time and computational resources, then the practical uses of such a controller are limited in an on-board application. To test the controller performance for this generalization, other references and other transfers are examined.

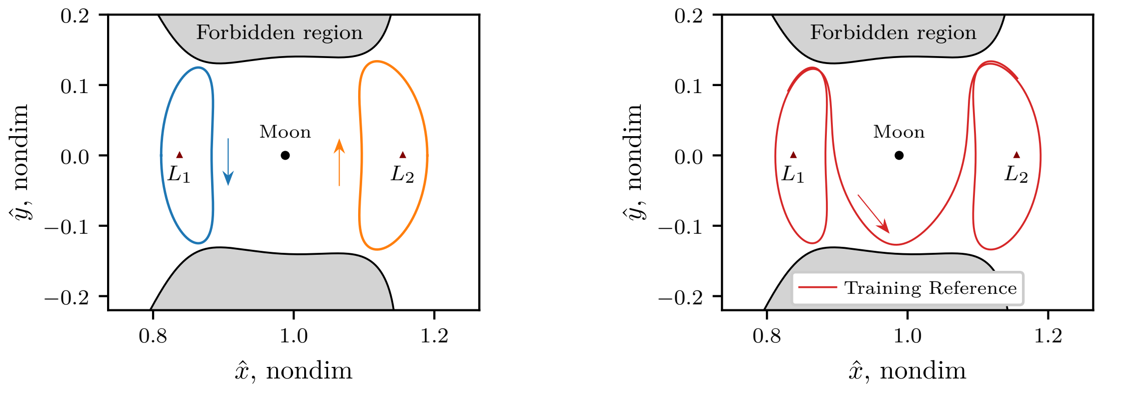
5.1 Training Process for a Sample Scenario

Heteroclinic transfers between L_1 and L_2 Lyapunov orbits at $C = 3.124102$ are employed to demonstrate the training process, as well as to evaluate the performance of the guidance framework. The transfer scenario is illustrated in Figure 5.1, where

Figure 5.1(a) depicts the periodic orbits, and Figure 5.1(b) illustrates a continuous heteroclinic transfer from the L_1 Lyapunov orbit to the L_2 Lyapunov orbit. The agent is trained using this heteroclinic transfer, labeled the “training reference”, and is subsequently evaluated using other transfer geometries.

The RL agent is trained over 120,000 finite-horizon episodes. During each episode, the agent attempts to maintain the training reference trajectory for as long as possible. When either the maximum number of time steps is reached, or the agent deviates too far from the reference, the episode is terminated, and then reset. Over the course of training, the agent's performance gradually improves until it consistently reaches the specified final time. Training performance over time is analyzed to understand the behavior of the RL algorithm at a high level, while specific examples lend insight into the practical applications of the controller.

During training, $\sigma_r = 300$ km and $\sigma_v = 4$ m/s model initial deviations from the reference. Without control, 1000 perturbed initial conditions are depicted in Figure 5.2(a), where each initial state is propagated for 17.4 days. The propagated initial perturbed states are plotted alongside the unstable manifold tube for the L_1



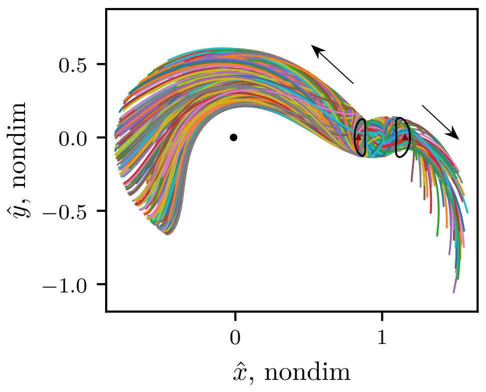
(a) L_1 and L_2 Lyapunov orbits. (b) Heteroclinic transfer employed in training.

Figure 5.1. Periodic orbits and heteroclinic transfer applied to the sample scenario. All motion at $C = 3.124102$, with the corresponding forbidden regions in gray.

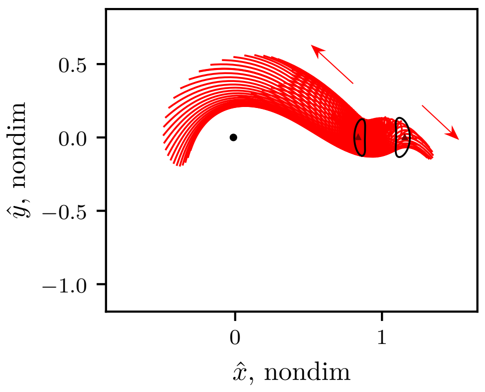
Lyapunov orbit. While the perturbed initial states are not on the manifold itself, they follow the geometry of the manifold tube, as plotted in Figure 5.2(b).

5.1.1 Training Process

After 120,000 episodes of training, a deterministic controller is produced. The user-specified hyperparameters in the environment and the RL algorithm are listed in Appendix A, Table A.1. During training, the total accumulated reward per episode is recorded to evaluate the training performance over time. The agent seeks to maximize the expected future discounted return, Eq. (2.5), at each time step. Therefore, the total accumulated reward per episode is expected to increase over the duration of training. For the sample scenario, a moving average of the reward sum per episode is plotted in Figure 5.3. The agent begins by randomly selecting actions and, at first, the agent is unable to accumulate reward. Then, once the agent begins learning an effective policy, reward sharply increases. For the sample agent, this sharp increase occurs approximately between episodes 8,000 and 15,000. After the initial surge in reward, performance improves gradually over the subsequent 75,000 episodes. Fi-



(a) 1000 perturbed initial states



(b) L_1 Lyapunov unstable manifold

Figure 5.2. Perturbed initial states and the unstable manifold tube for L_1 Lyapunov orbit depicted in Figure 5.1(a), all propagated for 17.4 days.

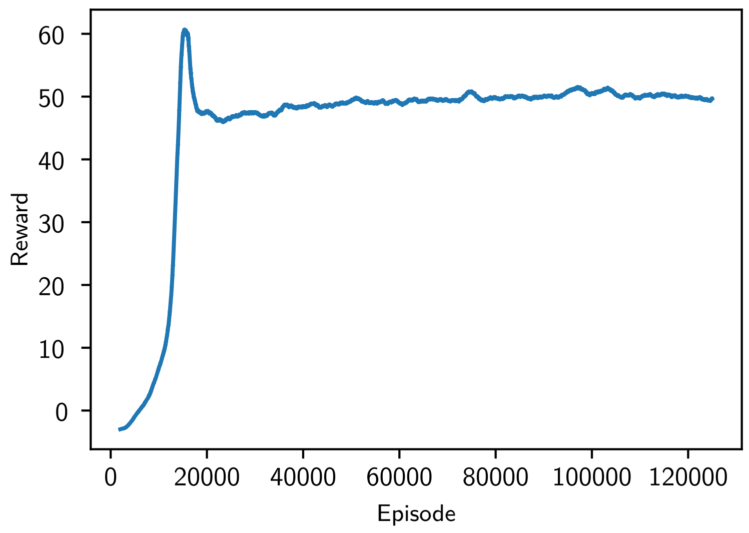
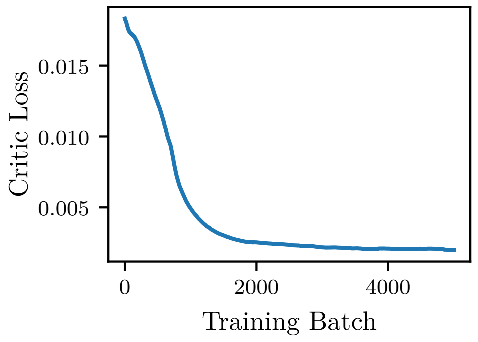


Figure 5.3. Moving average of total accumulated reward per episode for the sample scenario.

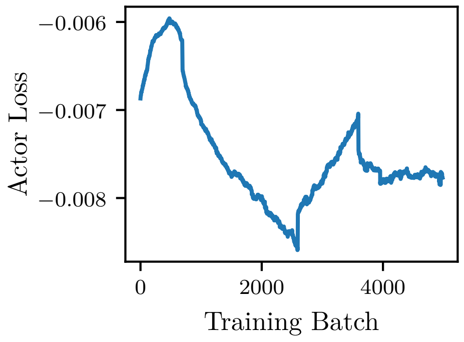
nally, as the agent approaches a deterministic policy, the reward curve reaches an approximate steady state, and the training is complete. In Figure 5.3, the moving average along the reward curve reaches a steady state value of approximately 49 around episode 90,000.

Once each batch is accumulated, the actor and critic networks are optimized over a finite number of training epochs. A rolling average for the mean loss function values over the training epochs for each batch are plotted in Figure 5.4. The critic steadily converges over the training process, which indicates that, upon completing training, the critic function accurately predicts the expected value of particular states. In contrast, the actor loss does not exhibit steady convergence toward zero. This is expected due to actor constantly discovering improved control characteristics as training progresses. However, the absence of convergence may indicate that further episodes could improve actor performance.

At any given point during training, a deterministic controller is available by simply removing variance from the agent's computed actions. Running an episode with the deterministic controller at various points in training, given a fixed initial condition,



(a) Critic Loss



(b) Actor Loss

Figure 5.4. Moving average of loss function values over each training batch for the actor and critic networks.

yields insight into the evolving improvement in the agent's policy. For this simulation, the initial conditions are generated by selecting an initial state along the L_1 Lyapunov departure orbit and introducing perturbations of 1106 km and 6.9 m/s. In reality, error is computed relative to the reference trajectory rather than the starting orbit, which can add a small amount of additional perturbation. For this sample case, the error relative to the reference is computed as 1108 km and 6.7 m/s. Without control, the resulting trajectory, as plotted in Figure 5.5, immediately deviates from the reference and impacts the Moon in less than a week. This introduced perturbation is not included at any point during the training phase. In many types of machine learning, it is important to separate training and validation data. In RL, this data separation is not as explicit as in supervised learning approaches, but it is nevertheless an important consideration when selecting a test case.

The control history produced by a deterministic controller at various stages in the training is plotted in Figure 5.6. Arrows indicate the thrust magnitude and direction for a particular segment, where the length and color of the arrow corresponds to thrust magnitude. For clarity, thrust values below a user-defined threshold cause the thrust direction magnitude indicator arrows to be omitted from visualization in Figure 5.6. In this case, 25% is arbitrarily defined. For practical applications, depending on

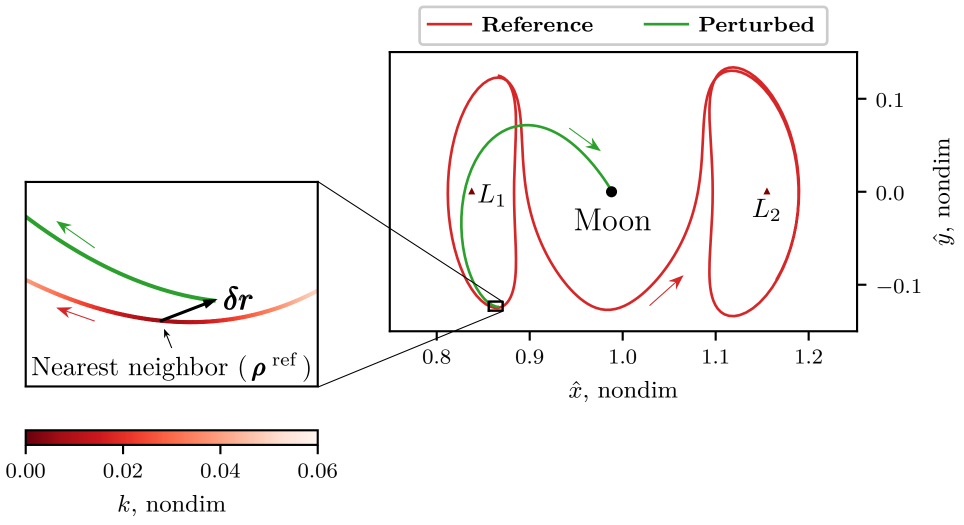
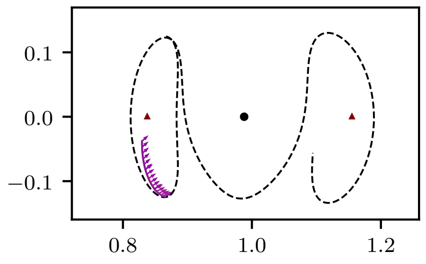


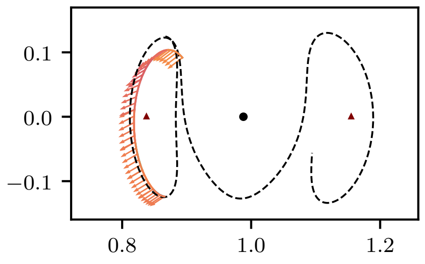
Figure 5.5. Sample perturbed initial state that impacts the Moon in 6.4 days. Position perturbation is plotted in configuration space ($\boldsymbol{\delta r}$), with magnitude 1106 km. States along the reference trajectory $\mathcal{R}^{\text{ref}}$ appear in the zoomed portion with the shade of red denoting the magnitude of the nearest neighbor distance to the perturbed state, k, defined in Eq. (4.3) (from LaFarge et al., Ref [43], Fig. 5).

the specific engine characteristics, this threshold possesses physical significance since low-thrust engines only deliver a thrust level within particular bounds. To eliminate the segments where f is small, a differential corrections algorithm could be applied with thrust and coast arcs delineated by the user-defined threshold, as detailed by Das-Stuart et al. [15].

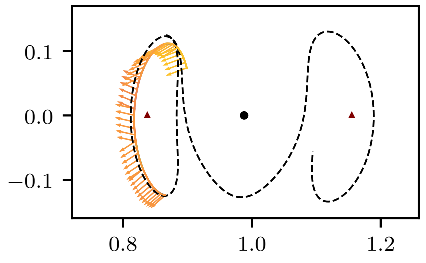
When training begins, the agent's policy is determined by randomly initialized weights in the actor neural network, depicted in Figure 2.4. In the sample case at episode 0, plotted in Figure 5.6(a), the agent's action choice worsens the perturbation and causes the episode to terminate quickly. As the number of episodes increases, the agent's ability to perform the given task gradually improves. By episode 15,000, Figure 5.6(d), the agent departs the Lyapunov orbit at approximately the correct location. After only 5,000 additional episodes, the agent completes the given transfer, though with substantially more thrusting than is necessary. Over the next 25,000 episodes, the agent gradually improves by reducing the amount of thrust applied. To



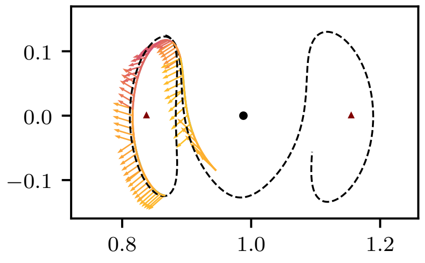
(a) 0 Episodes: -3.995 reward



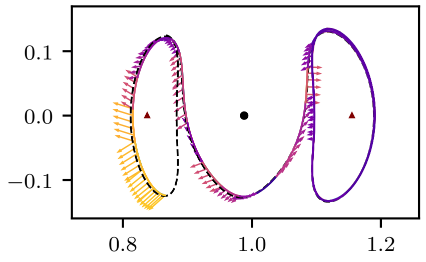
(b) 5,000 Episodes: -3.73 reward



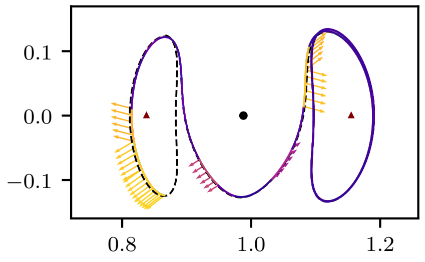
(c) 10,000 Episodes: -3.57 reward



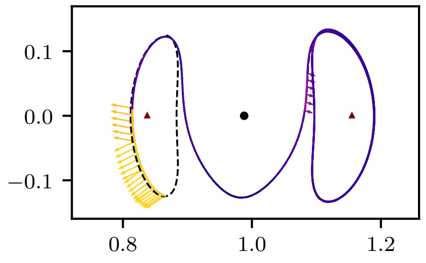
(d) 15,000 Episodes: -2.61 reward



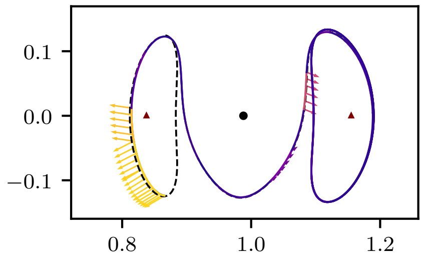
(e) 20,000 Episodes: 212.85 reward



(f) 40,000 Episodes: 258.0 reward



(g) 80,000 Episodes: 545.7 reward



(h) 120,000 Episodes: 983.6 reward

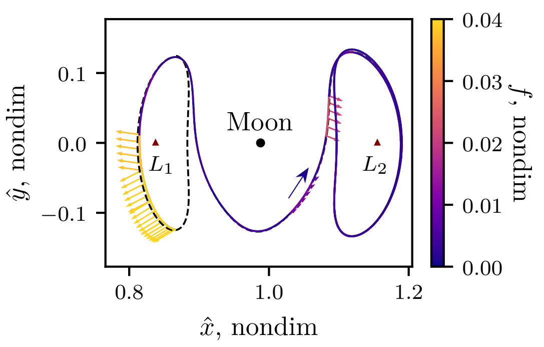
Figure 5.6. Deterministic agent over 120,000 episodes ($\hat{x}$-$\hat{y}$, nondim)

illustrate the thrust reduction over learning, at episode 20,000, the agent performs the given task, but expends 0.69% of the available propellant. By episode 80,000, Figure 5.6(g), expended propellant is reduced to 0.34%. Furthermore, this thrust reduction is visually apparent in the departure segment from the L_1 Lyapunov orbit.

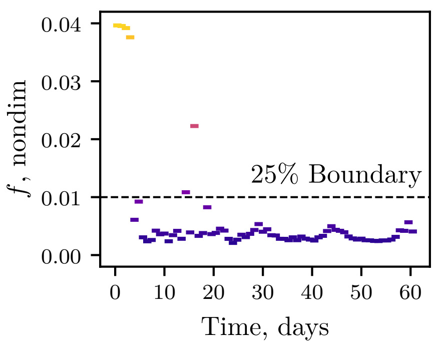
5.2 Closed-loop Controller Performance

With training complete, the performance of the resulting deterministic controller is analyzed separately from the training process. For the perturbation in Figure 5.5, the sample controller produces the control history in Figure 5.7. The trajectory in Figure 5.7(a) is identical to Figure 5.6(h) since the resulting controller is simply the deterministic controller at the final episode. Given the introduction of the perturbation, with the goal of returning to the original reference trajectory, a delay in response time renders the original geometry inaccessible. However, the trained neural network controller immediately outputs a control history that returns the spacecraft to its original path and successfully accesses the target L_2 Lyapunov orbit. As expected, the majority of the spacecraft's thrusting occurs during the initial time intervals as the agent recovers from the initial perturbation. Subsequent time intervals require much less propellant. Upon arrival in the destination L_2 Lyapunov orbit, the controller maintains the arrival geometry. The orbit maintenance is apparent in configuration space, i.e., in Figure 5.7(a), as well as in the periodic sinusoidal thrust magnitude behavior, depicted in Figure 5.7(b). Again, thrust magnitudes below 25% are omitted from visualization, but are still applied in the dynamical model.

To analyze the performance of a particular controller, many initial conditions with various perturbations are generated and the agent is evaluated based on its resulting output control history. A simulation is considered successful if the agent avoids deviating from the reference and reaches a specified arrival criteria. 'Arrival' occurs when the relative position and velocity magnitudes relative to a state along the arrival orbit are less than 100 km and 2 m/s, respectively. This definition of arrival is not intended to indicate that the spacecraft has reached a particular state. Instead,



(a) Trajectory and control history.



(b) Thrust magnitude history.

Figure 5.7. Sample case where controller successfully overcomes an initial perturbation and follows a given reference trajectory. Thrust direction and magnitude are plotted in configuration space and colored based on thrust magnitude. For thrust values below an arbitrary threshold (25%), thrust direction indicators are omitted from the control history in (a).

the given tolerances simply detect that the spacecraft has reached the vicinity of the arrival orbit. If a tighter tolerance for arrival is employed, such as the values demonstrated in LaFarge et al. [43], many false negatives exist that eventually reach the specified tolerance, but not within the specified maximum number of time steps.

The numerically computed arrival percentages for various levels of 3σ error in position and velocity are depicted in Figure 5.8. For 10,000 combinations of position and velocity 3σ values, 1000 deterministic episodes are simulated using the sample agent and the training reference trajectory. The results of the Monte Carlo analysis are colored based on the arrival percentage across the 1000 episodes. Based on expected levels of position and velocity error, Figure 5.8 demonstrates the expectations for controller performance. For example, if all position and velocity errors are expected to be less than 1000 km and 10 m/s, respectively, then the bright yellow color at $[3\sigma_v = 10 \text{ m/s}, 3\sigma_r = 1000 \text{ km}]$ indicates that the controller is expected to reach the destination orbit in nearly 100% of cases (actual value is 99.4%). As expected, as the error increases, the controller becomes less successful. With large amounts of error, it is frequently unreasonable to return to a previous reference trajectory. Cases where

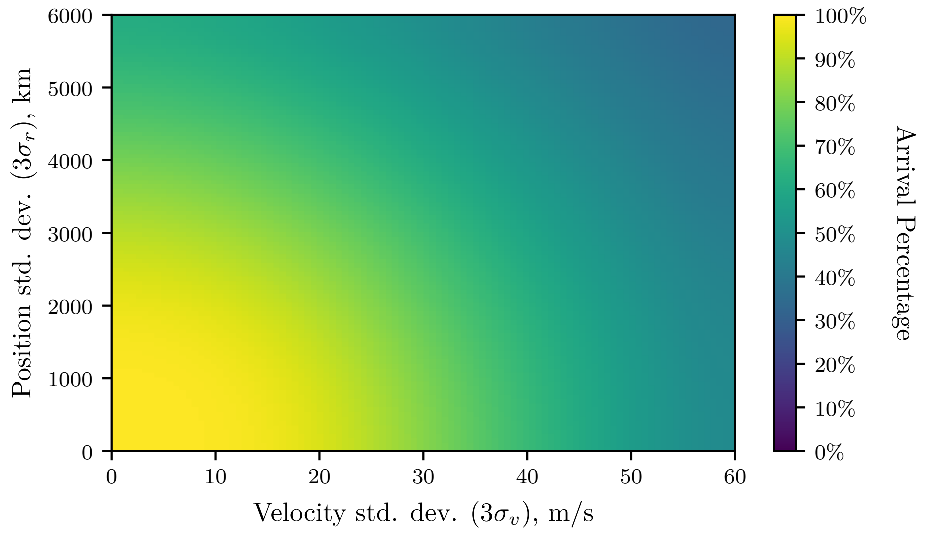


Figure 5.8. Arrival percentage for training reference given various 3σ levels of initial error. Each error combination is simulated for 1000 deterministic episodes, and the resulting data is displayed as a colormap, with a Guassian blur applied to reduce image noise.

the controller frequently fails to recover indicate error levels where a new reference trajectory is required.

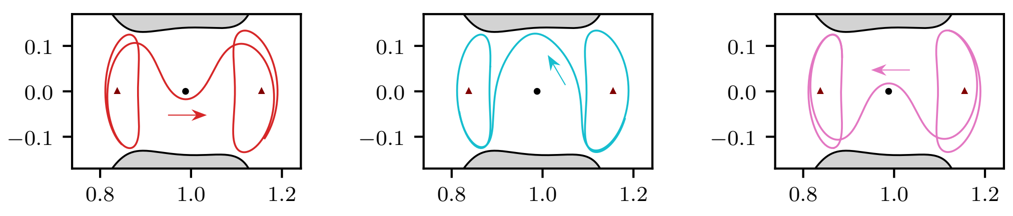
The sample controller is more sensitive to perturbations in velocity than position, as depicted in Figure 5.8. With perturbations sampled from $3\sigma_v = 0$ m/s and $3\sigma_r = 6000$ km, the controller arrives successfully in 59.1% of the cases. Conversely, if $3\sigma_v = 60$ m/s and $3\sigma_r = 0$ km, the agent succeeds in only 44.1% of the cases. This sensitivity to velocity errors is consistent with other applications in cislunar space. For example, Davis et al. demonstrate the significant impact of velocity navigation errors in NRHO orbit maintenance [69].

5.2.1 Generalization to Other References

If a spacecraft deviates significantly from its reference path, it is not always reasonable to return to the original trajectory. The process of generating a new transfer arc is accomplished from a variety of options, including leveraging dynamical systems theory, applying a numerical strategy, and/or employing differential corrections. The methodology for generating new reference transfers is not considered in this investigation. However, assuming a new trajectory has been constructed, an important test of the proposed controller is its ability to perform despite training with only one original path. While neural networks, in general, offer a demonstrated ability to generalize, their performance is always tied to the training data set. Hence, if the new geometry is vastly different, a NN-based approach is limited by its training experience.

To test the extendability of the PPO-generated controller, several new transfers are examined. First, a new path between the original L_1 and L_2 Lyapunov orbits is demonstrated via a second heteroclinic reference trajectory in Figure 5.9(a). Next, the reverse of the previous example is also included, in which the agent starts in the L_2 Lyapunov orbit and attempts to track heteroclinic transfers to the L_1 Lyapunov orbit. These two new heteroclinic paths are plotted in Figure. 5.9(b) and Figure. 5.9(c).

To evaluate the scenario where a new reference is generated in-flight, Reference 2 in Figure 5.9(a) is employed. Reference 2 is not included in the training process and, therefore, the controller must extrapolate its experience on the training reference to a different area of cislunar space. The new path is notably different from the original one; the nearest distance to the Moon along the training reference and Reference 2 are 34,546 km and 6,725 km, respectively. Not only is this a large deviation in



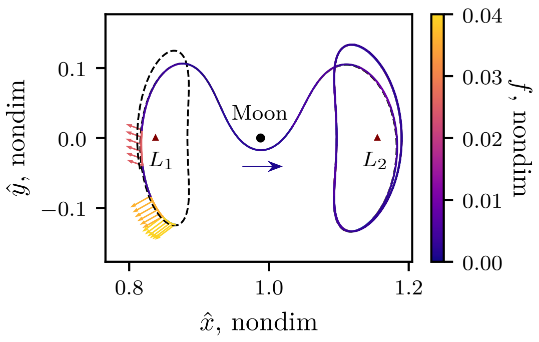
(a) Reference 2 (L_1 to L_2). (b) Reference 3 (L_2 to L_1). (c) Reference 4 (L_2 to L_1).

Figure 5.9. Additional heteroclinic transfers between Lyapunov orbits at $C = 3.124102$ ($\hat{x}$-$\hat{y}$, nondim). Reference 2 (a) connects L_1 to L_2, and passes closer to the Moon than the training reference. References 3 (b) and 4 (c) connect L_2 to L_1, and are the mirror images of the training reference and Reference 2.

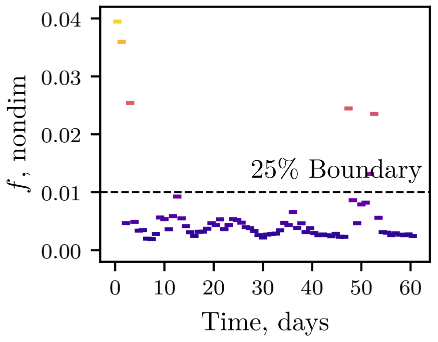
geometry, significant nonlinearity is added due to the close proximity with the Moon. Hence, this example adds difficulty for the controller, not only in extrapolating to new locations in space, but also demanding that the controller overcome more nonlinearity than was present in the training. Returning to the previous example, the perturbation in Figure 5.5 is applied to the new reference geometry. The error from the perturbed state to its nearest neighbor along the new reference path is 860 km and 5.1 m/s. The resulting control history and successful transfer are plotted in Figure 5.10. The agent is clearly able to extrapolate to the new reference by generalizing its experience given the relative state information.

Reverse Transfer Scenario

For realistic scenarios, since the agent's performance is dependent on training data, it is generally inadvisable to reuse an old agent for a drastically different scenario without training on the new geometry. However, examples where no additional training is conducted are included to illustrate the agent's ability to perform well in regions of space that were not originally explored. In particular, the environment is reversed from the previous example, that is, the agent is required to transfer from



(a) Trajectory and control history.



(b) Thrust magnitude history.

Figure 5.10. Resulting control history for previous sample deviation, plotted in Figure 5.5, but using Reference 2 from Figure 5.9(a). Without additional training, the agent successfully tracks a new reference trajectory.

the L_2 Lyapunov orbit to the L_1 Lyapunov orbit along one of the heteroclinic paths plotted in Figure. 5.9(b) and Figure. 5.9(c).

A single sample case is again examined. For a perturbation illustrated in Figure 5.11, the uncontrolled path departs the vicinity of the Moon. Again, the agent is tasked to return to one of the new references, without the benefit of previous training. The controller completes these new transfers, with the resulting control histories plotted in Figure 5.12. These examples demonstrate the RL controller's ability to perform well in challenging scenarios.

Generalization Limitations

While the controller demonstrates a remarkable ability in generalizing to new reference trajectories, there are several notable scenarios in which the controller is unable to generate an effective control strategy. The sample controller is effective in the vicinity of the Moon. However, for heteroclinic transfers that pass through the L_1 or L_2 portal, the agent produces poor control choices, and eventually deviates from the new reference. Two examples of the controller deviating are plotted in Figure 5.13. Given two heteroclinic transfers at the same energy level as the training reference, the agent fails on both an interior transfer, Figure 5.13(a), and an exterior transfer, Figure 5.13(b). These failures indicate that, if a drastic change in the reference geometry is possible, the training process for the controller must reflect all possible regimes in space that the controller may encounter.

Another notable sensitivity in the controller is changes in energy. Slightly decreasing the energy of the training reference and the Lyapunov orbits, increasing Jacobi constant from $C = 3.124102$ to $C = 3.13$, produces a new transfer scenario depicted in Figure 5.14(a). While the transfer exhibits similar geometry in configuration space to the training reference, the controller is unable to consistently execute this lower energy transfer. An example of a failure case is plotted in Figure 5.14(b). This limitation indicates that the agent learns the dynamics associated with a particular energy

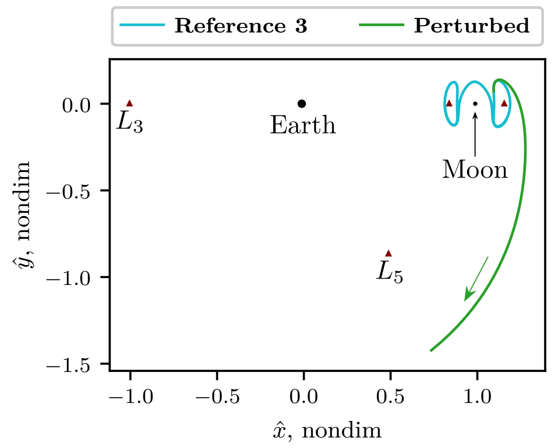
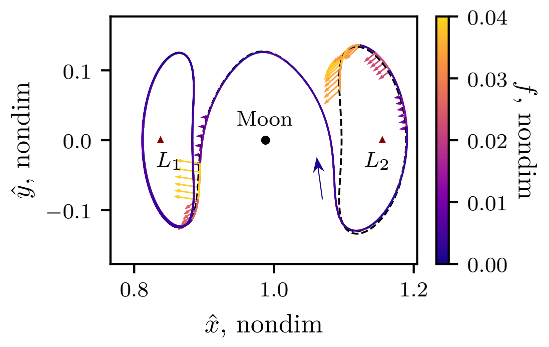
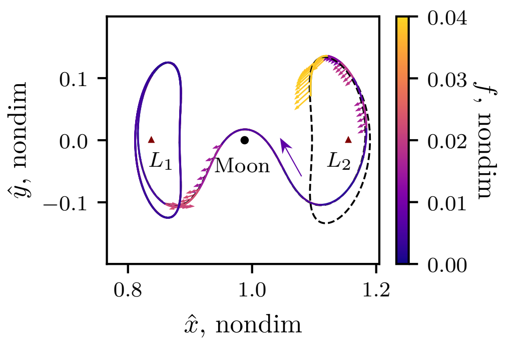


Figure 5.11. Sample perturbation from L_2 Lyapunov orbit for evaluating agent's ability to control a reference trajectory in a different direction (from LaFarge et al., Ref [43], Fig. 8(b)).

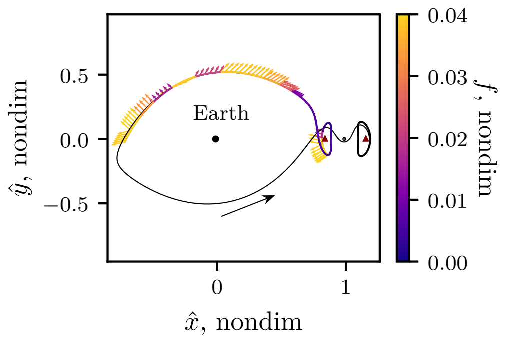


(a) Control history for Reference 3

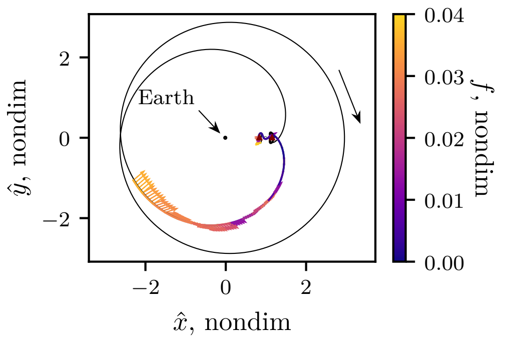


(b) Control history for Reference 4

Figure 5.12. Computed control histories for the L_2 to L_1 reference trajectories plotted in Figure 5.9(b) and Figure 5.9(c), given the perturbation depicted in Figure 5.11.

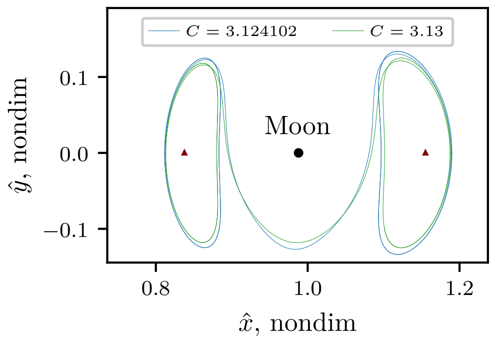


(a) Interior Heteroclinic Transfer.

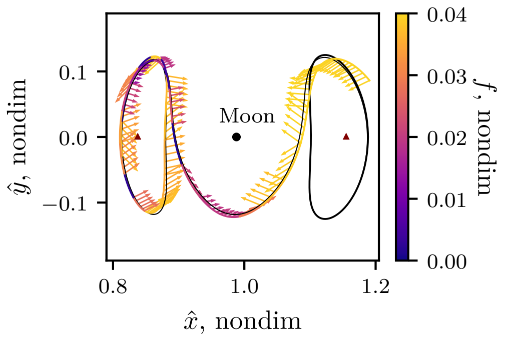


(b) Exterior Heteroclinic Transfer.

Figure 5.13. Failures to generalize to new references located in different regions of cislunar space, but at the same Jacobi constant value as the training reference.



(a) Transfers at two energy levels



(b) Sample episode

Figure 5.14. Failure to generalize to a new reference trajectory at a slightly different energy level.

level. If energy changes are likely for a specific scenario, then the training episodes would need to incorporate variations in energy.

5.2.2 Monte Carlo Results

A Monte Carlo analysis approach is applied to evaluate the sample controller's performance across different references. For this analysis, multiple levels of error are simulated for 50,000 trials each across the four sample reference trajectories.

Assuming that orbit determination navigation errors are on the order of $3\sigma = 1$ km and 1 cm/s [65], the proposed controller is tested with errors up to 2000 times the expected navigation error. The 1000× case roughly corresponds to the error used in training, which represents the situation where a 1 km and 1 cm/s perturbation is introduced, and the error is propagated for an orbital period, or about 12.9 days.

The results from the Monte Carlo analysis is summarized in Table 5.3, with failure rates plotted in Figure 5.15. For the training reference, the controller is 100% successful for all errors less than the 500× scenario. However, as error is increased, trials emerge where the controller is unable to recover. In the 1000× case, 0.6% of perturbations are not successfully recovered. For some of these examples, where the error bounds are $3\sigma = 1000$ km and 10 m/s, it is unreasonable to expect a return to the original motion, and these error levels may correspond to conditions where alternate options should be considered. Four common causes of failure are listed in Table 5.4.

To illustrate the failure characteristics that occur for the 1000× case, deviations over time for 100 sample episodes are represented in Figure 5.16. The red trajectories correspond to failures where the episode is terminated when either of the maximum deviation thresholds is violated, or if arrival conditions are not met within the maximum number of time steps. Episodes that reach the arrival criteria, in green, clearly maintain a small amount of error upon arrival to the target orbit. However, during the transfer phase, more variance is observed in the deviations. This variation is likely caused by selection of a sub-optimal action which forces the agent to recover from additional error.

The arrival percentages for References 2–4 further demonstrate the agent's ability to generalize experience from the training reference to the other three geometries. For Reference 2, failure rates are consistently $\approx 1\%$ less than the training reference. However, given the large amount of error, the agent still arrives 98.4% of the time in the 1000× case. This high-performance level demonstrates the neural network controller's ability to perform well despite significant uncertainty. For Reference 3,

Table 5.3. Arrival percentages for Monte Carlo simulations given various reference trajectories with 50,000 deterministic episodes each. The agent is only trained using the training reference, and extrapolates to References 2, 3, and 4 (Figure 5.9). The 1000× case corresponds to the amount of error the agent experiences during training.

	L_1 to L_2		L_2 to L_1	
Error Amount	Training Reference	Reference 2	Reference 3	Reference 4
1×	100%	99.3%	99.6%	97.4%
10×	100%	99.4%	99.7%	97.4%
100×	100%	99.3%	99.6%	97.6%
1000×	99.4%	98.4%	97.7%	97.6%
2000×	88.0%	86.8%	85.1%	90.0%

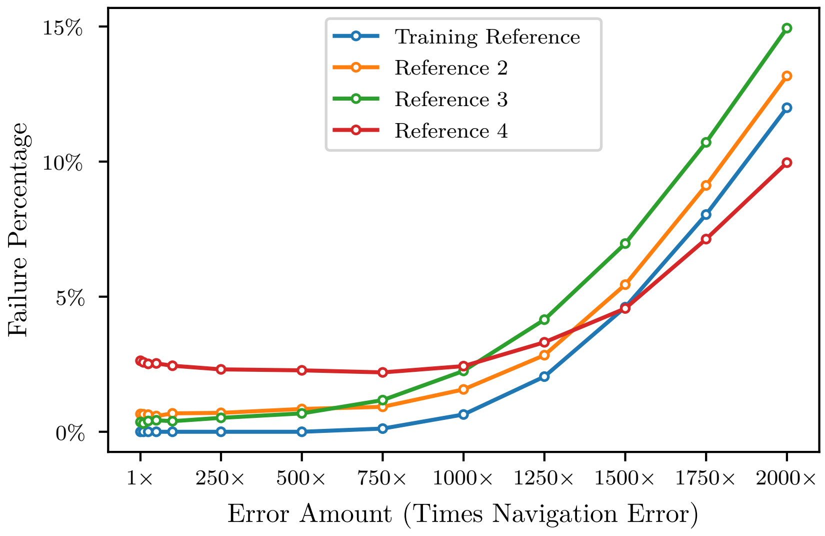


Figure 5.15. Failure rates for Monte Carlo simulations given the four reference trajectories with 50,000 deterministic episodes each. Performance remains mostly level until the error amount exceeds the 1000× threshold.

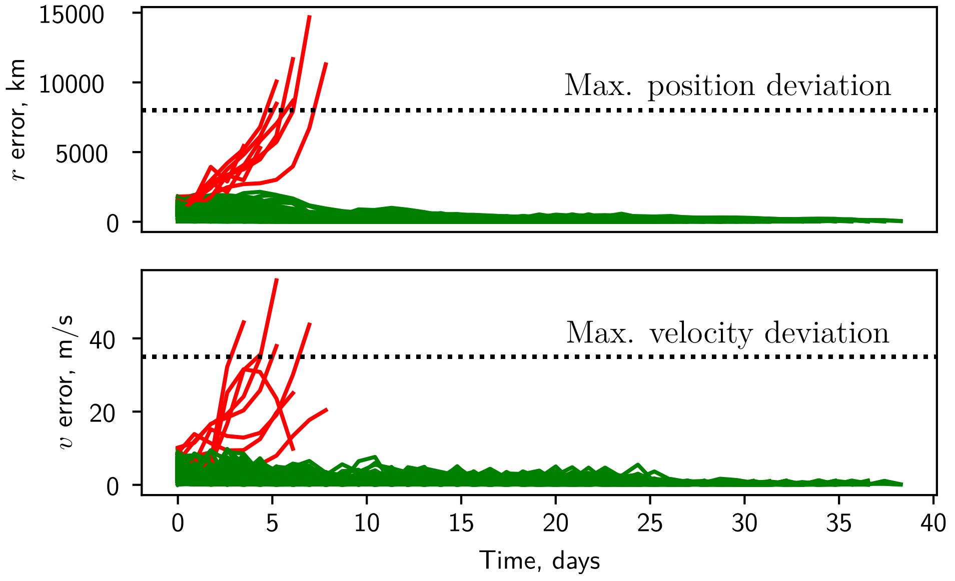


Figure 5.16. Deviation over time for 1000 sample episodes using the training reference. The initial perturbation variance 3σ is 1000 times greater than the expected navigation error. Green denotes episodes that reach the success criteria, whereas red signifies trajectories that cross the maximum deviation threshold in position or velocity, or do not reach arrival conditions within the maximum number of time steps (130.46 days). In the trials, 992/1000 episodes are deemed successful for this sample batch of initial conditions.

where the spacecraft does not approach the Moon, the agent performs exceedingly well, reaching the arrival criteria more that 97% of the time for error levels up to 1000×. Furthermore, when the agent is required to fly nearby the Moon along Reference 4, it is still successful in more than 97% of the scenarios for error levels up to 1000×. In consistently satisfying the arrival criteria for this new scenario, the agent demonstrates a remarkable ability to extrapolate to new geometries. These results could potentially be improved by training the existing agent with the new transfers, i.e., employing transfer learning with the controller, but this possibility was not investigated.

Table 5.3 also demonstrates the controller's robustness to perturbation levels. Despite the three orders of magnitude difference between error distributions for 1×

and 1000×, results remain within 2% for each reference. During training, the agent experiences 1000× error levels and, hence, learns to overcome them. When error is further increased, performance degrades. However, for values up to those used in training, performance only slightly decrease for 3/4 references, and does not degrade for Reference 4.

5.3 Algorithm Limitations

Reinforcement learning methods are often inconsistent and difficult to reproduce [70]. Small changes in implementation and hyperparameters can yield varying results. Furthermore, even with identically configured learning schemes, a wide variety of resulting controllers is produced. The stochasticity of the training process produces a nonzero probability of the agent getting 'stuck' in a local basin of attraction and failing to converge to a better policy. Hence, multiple identically configured agents often converge to drastically different policies depending on their exploration of the action space. This inconsistency in training renders a difficulty with reproducibility because many agents must be simulated to discover one that achieves the desired result. The training process is computationally demanding, thus, a large number of simulations are challenging. This investigation addresses this limitation by leveraging supercomputing resources, via MIT Supercloud [71], to train many agents simultaneously. Once trained, each agent is tested in a set of deterministic episodes, and the accrued reward and arrival percentages from these trials allow the extraction of desirable agents.

Training many identically configured agents in parallel offers insight into the variation in controller performance produced by the RL scheme. Figure 5.17 illustrates the distribution in arrival percentage on the training reference for the deterministic controller produced by 347 agents with the identical hyperparemeters listed in Table A.1. In this simulation, the majority (55.3%) of the agents perform well and reach the arrival criteria in at least 95% of cases. However, more than 25% of deterministic controllers fail at least 40% of the time. With computationally expensive training, this inconsistency makes producing a desirable agent difficult.

To analyze generalizability, controllers are also tested on the additional reference trajectories and transfer scenarios summarized in Figure 5.9. Histograms of the arrival percentages along three additional transfer scenarios for the 347 agents are depicted in Figure 5.18. While the RL scheme produces many controllers that perform well on the training reference, there is more variability in generalization performance. In particular, for Reference 3, only 21% of controllers arrive for at least 90% of cases, and only 10% reach the 95% threshold. Furthermore, some controllers only effectively generalize to one or two additional references. Of the 347 trained agents, only three reach 95% arrival on all four transfer scenarios. One of these three is employed as the sample controller in this investigation.

Examining the reward curves for the 347 agents also yields insight into variations in training. Recall that the sample agent's reward curve, Figure 5.3, exhibits a sharp initial increase, followed by a gradual leveling of the accumulated reward. In contrast, the reward curves for thirteen identically configured agents are plotted in

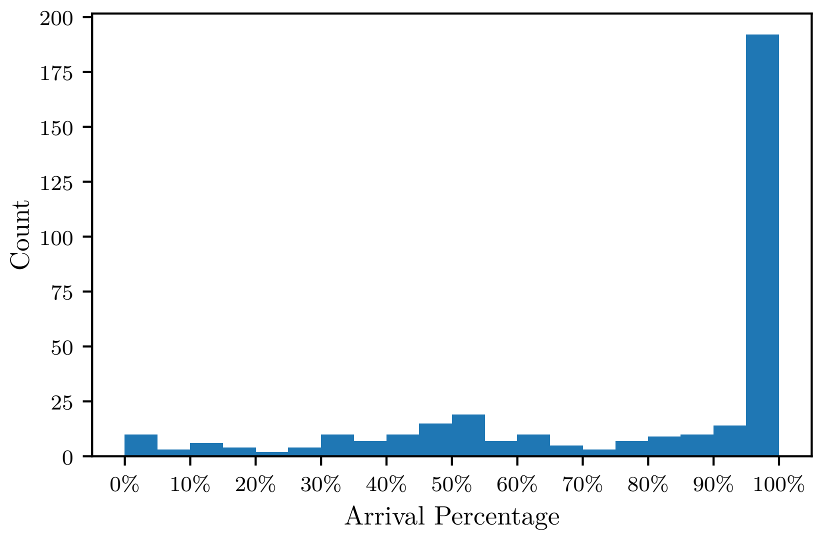


Figure 5.17. Arrival distribution for the training reference

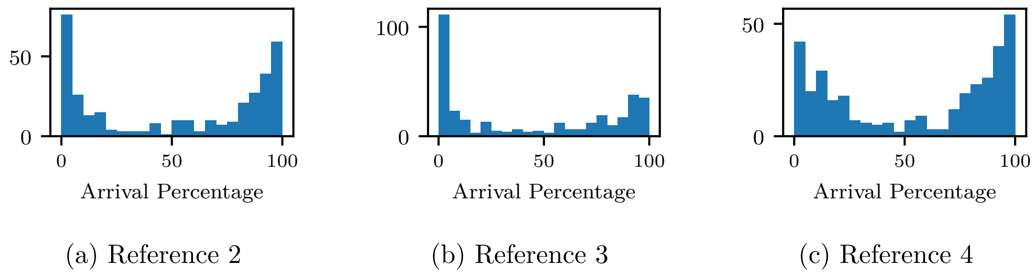


Figure 5.18. Arrival distributions for additional reference trajectories not included in training. Each trajectory is plotted in Figure 5.9, and percentages are computed based on 1000 simulations of the deterministic controller.

Figure 5.19. While several exhibit similar behavior as the sample controller, there are many that do not follow the same trend. This variation stems from differences in the stochastic exploration and illustrate the impact of different local basins producing different training progress.

For the trained agent in the sample scenarios, several sources of failure are present with increased noise. First, for cases where the agent failed to arrive along the training reference, there are regions of ambiguity that cause a sub-optimal action

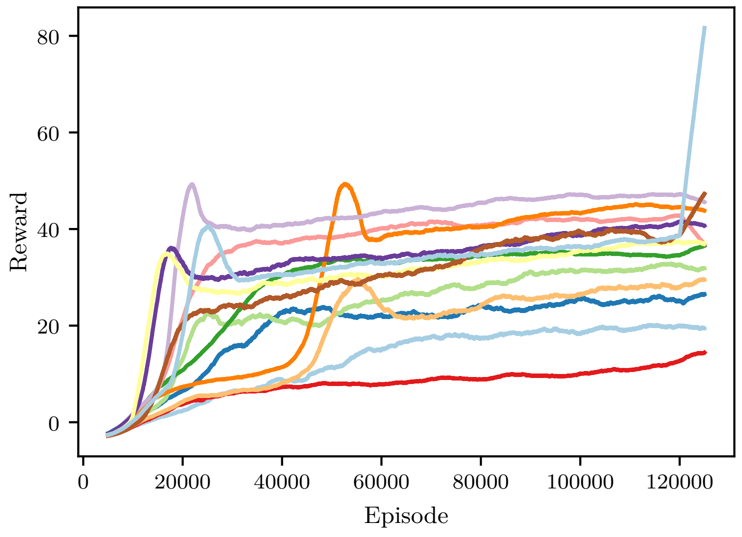
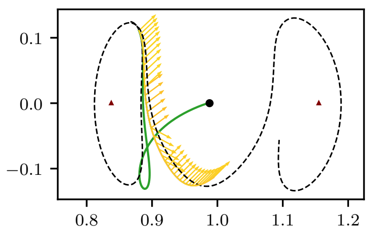
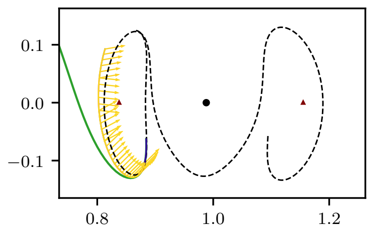
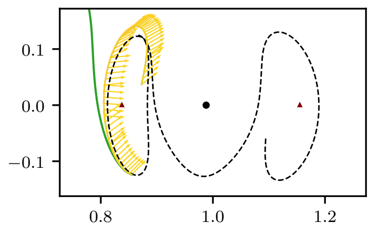
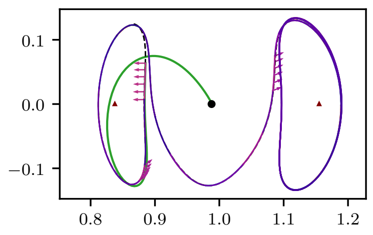


Figure 5.19. Reward curves for thirteen identically configured agents.

to produce an unrecoverable trajectory. An example of this behavior is plotted in the first row of Table 5.4. Here, the agent attempts to prematurely depart the L_1 Lyapunov orbit, which results in an unrecoverable scenario. This ambiguity in thrust direction occurs in regions where two different segments of the reference path are nearby. In these regions, the agent is more likely to implement a poor action since, frequently, only one of the nearby segments is actually accessible. This incorrect thrust direction also demonstrates the importance of the initial action. Due to the introduced perturbation, the first action is critical to performance. An example of a poor initial action yielding an unrecoverable error is depicted in the second row of Table 5.4. In this case, thrust is needed to recover from the perturbation, but the agent incorrectly implements a nearly-coasting arc that ensures the spacecraft departs the lunar vicinity. The explicit error cause in such examples is challenging to diagnose due to the ‘black box’ nature of neural networks.

Table 5.4. Noted sources of failure. In the sample control histories, green denotes the propagated uncontrolled perturbation (from LaFarge et al., Ref [43], Table 3).

Failure type	Sample control history ($\hat{x}$-$\hat{y}$, nondim)	Notes
Ambiguous thrust direction		Agent attempts to prematurely depart the L_1 vicinity instead of completing one more revolution around the orbit.
Poor initial action		After a poor initial action, the deviation becomes unrecoverable with the given maximum thrust level.
Unrecoverable perturbation		Perturbation is such that the agent departs the vicinity of the Moon regardless of action choice.
Arrival condition not triggered		Agent completes the transfer, but does not reach the defined arrival criteria within the maximum number of time steps.

6. CONCLUDING REMARKS AND FUTURE WORK

6.1 Concluding Remarks

Computationally efficient on-board guidance is challenging in nonlinear dynamical regimes. The proposed guidance framework addresses the challenges associated with automation given limited computational resources by recasting the problem from a machine learning perspective. The demonstrated controller offers computationally efficient on-board guidance in a multi-body regime. By decoupling training from the controller output, this approach utilizes high-performance computers while still producing an algorithm suited to the closed-loop flight environment. The resulting neural network controller is robust to changes in reference geometry, and generalizes past experience to new problems. Furthermore, the proposed approach separates the learning agent from the dynamical environment, enabling model-agnostic guidance that is extendable to higher-fidelity domains. In summary, the primary conclusions of this research investigation are as follows:

1. A controller trained via reinforcement learning overcomes perturbations and nonlinearity to autonomously compute a control history for a low-thrust spacecraft. In sample simulations, assuming perturbations three orders of magnitude greater than navigation errors, the controller successfully completes the given transfer in more than 99% of cases.

2. Neural networks serve as computationally efficient controllers that are potentially well-suited for the flight environment. Many current guidance methodologies necessitate access to a high-performance computer and would not be suitable for a flight computer. Conversely, many on-board approaches are computationally efficient, but unable to take advantage of ground-based supercomputing resources. With RL, the computationally intensive training process leverages

ground-based high performance computing, while producing a computationally efficient closed-loop controller.

3. Sample applications demonstrate remarkable controller reconfigurability and generalization. An RL-trained controller adapts to nearby geometries not encountered during training. This generalization ability indicates that the neural network controller continues performing in real-time despite shifts in mission objectives.

4. Reinforcement learning approaches all separate the agent from the environment, which makes learning schemes applicable to many dynamical realms. The demonstrated success of the controller in the planar CR3BP suggests that a PPO-trained controller may excel in higher-fidelity models, however this potential has not yet been investigated.

Transfers between libration point orbits support the stated conclusions and suggest the concept that RL-trained controllers can be effective in challenging dynamical regions of space.

6.2 Recommendations for Future Work

This research effort is a preliminary investigation into leveraging reinforcement learning to train a closed-loop controller. The sample application shows promise, and several avenues for future research are available.

- An attractive feature of RL is the separation between agent and environment. The learning scheme is agnostic to the dynamical model and, thus, is applicable to multiple domains. The current work leverages the planar CR3BP to demonstrate controller effectiveness within the context of a challenging dynamical model. There are many research avenues for applying the methods of this investigation to other dynamical environments, e.g., spatial CR3BP, bicircular restricted four-body problem, ephemeris, and small body operations.

- The current investigation leverages Proximal Policy Optimization (PPO) to train a controller. There are many other RL methods available that warrant exploration and may prove more productive than PPO. In particular, algorithms related to Deep Policy Gradient (DPG) methods, such as Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3), may prove productive. Furthermore, variations within PPO may improve performance, e.g., applying meta-learning by including a recurrent layer in the neural network architecture.

- The controller in this investigation directly computes a control history for the spacecraft. However, there are opportunities for hybrid approaches that use an RL-generated controller in conjunction with traditional techniques. In particular, the neural network controller may offer initial guesses for control variables in on-board targeting algorithms.

- This investigation leverages heteroclinic transfers as sample applications to demonstrate the controller. Further research is available in extending analysis to other types of transfers, impulsive engines, and stationkeeping applications.

- Other authors demonstrate RL's ability to include practical limitations and failure modes directly into the learning process [40]. The current analysis may be extended to include practical considerations, e.g., varying initial mass, modeling partial engine failures, or including maneuver execution errors.

Building on the demonstrated results in this work, future research avenues present exciting opportunities for advancing RL-enabled spacecraft guidance.

REFERENCES

[1] Ryan Whitley and Roland Martinez. Options for staging orbits in cislunar space. In *2016 IEEE Aerospace Conference*, pages 1–9, Big Sky, Montana, March 2016. IEEE.

[2] Jeremy Hart, Ellis King, Piero Miotto, and Sungyung Lim. Orion gn&c architecture for increased spacecraft automation and autonomy capabilities. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, pages 1–26, Honolulu, Hawaii, August 2008. AIAA.

[3] Richard S. Sutton and Andrew G. Barto. *Reinfrocement Learning: An Introduction.* The MIT Press, second edition, 2018.

[4] Amanda F. Haapala and Kathleen C. Howell. A framework for constructing transfers linking periodic libration point orbits in the spatial circular restricted three-body problem. *International Journal of Bifurcations and Chaos*, 26(5), 2016.

[5] Melissa McGuire, Laura Burke, Steven McCarty, Kurt J Hack, Ryan Whitley, Diane C Davis, and Cesar Ocampo. Low thrust cis-lunar transfers using a 40 kw-class solar electric propulsion spacecraft. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–21, Stevenson, Washington, August 2017. American Astronautical Society.

[6] Matthew A. Vavrina, Jacob A. Englander, Sean M. Phillips, and Kyle M. Hughes. Global, multi-objective trajectory optimization with parametric spreading. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–20, Stevenson, Washington, August 2017. American Astronautical Society.

[7] Belinda G. Marchand, Michael W. Weeks, Chad W. Smith, and Sara Scarritt. Onboard autonomous targeting for the trans-earth phase of orion. *Journal of Guidance, Control, and Dynamics*, 33(3):943–956, 2010.

[8] Michael I Jordan. Artificial intelligence – the revolution hasn't happened yet. Medium, Apr 2018. [Online; accessed 19-March-2020].

[9] Tara A Estlin, Benjamin J Bornstein, Daniel M Gaines, Robert C Anderson, David R Thompson, Michael Burl, Rebecca Castaño, and Michele Judd. Aegis automated science targeting for the mer opportunity rover. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):1–19, 2012.

[10] R Francis, T Estlin, G Doran, S Johnstone, D Gaines, V Verma, M Burl, J Frydenvang, S Montano, RC Wiens, S Schaffer, O Gasnault, L Deflores, D Blaney, and B Bornstein. Aegis autonomous targeting for chemcam on mars science laboratory: Deployment and results of initial science team use. *Science Robotics*, 2(7), 2017.

[11] Bernd Dachwald. Evolutionary neurocontrol: A smart method for global optimization of low-thrust trajectories. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, pages 1–16, Providence, Rhode Island, August 2004.

[12] Stijn De Smet and Daniel J. Scheeres. Identifying heteroclinic connections using artificial neural networks. *Acta Astronautica*, 161:192–199, August 2019.

[13] Nathan L. Parrish and Daniel J. Scheeres. Optimal low-thrust trajectory correction with neural networks. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–20, Snowbird, Utah, 2018. American Astronautical Society.

[14] Roberto Furfaro, Ilaria Bloise, Marcello Orlandelli, Pierluigi Di Lizia, Francesco Tupputo, and Richard Linares. Deep learning for autonomous lunar landing. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–22, Snowbird, Utah, 2018. American Astronautical Society.

[15] Ashwati Das-Stuart, Kathleen C Howell, and David C. Folta. Rapid trajectory design in complex environments enabled by reinforcement learning and graph search strategies. *Acta Astronautica*, 171:172–195, 2020.

[16] Lena M. Downes, Ted J. Steiner, and Jonathan P. How. Deep learning crater detection for lunar terrain relative navigation. In *20th AIAA Scitech Forum*, Orlando, Florida, January 2020. AIAA.

[17] G. K. Benedix, C. J. Norman, P. A. Bland, M.C. Towner, J. Paxman, and T. Tan. Automated detection of martian craters using a convolutional neural network. In *49th Lunar and Planetary Science Conference*, The Woodlands, Texas, March 2018.

[18] Ryan Alimo, Daniel Jeong, and Kingson Man. Explainable non-cooperative spacecraft pose estimation using convolutional neural networks. In *20th AIAA Scitech Forum*, Orlando, Florida, January 2020. AIAA.

[19] Lorenzo Pasqualetto Cassinis, Robert Fonod, Eberhard Gill, Ingo Ahrns, and Jesus Gil Fernandez. Cnn-based pose estimation system for close-proximity operations around uncooperative spacecraft. In *20th AIAA Scitech Forum*, Orlando, Florida, January 2020. AIAA.

[20] Mar Vaquero and Juan Senent. Poincare : A multibody, multi-system trajectory design tool. In *7th International Conference on Astrodynamics Tools and Techniques*, pages 1–12, Oberpfaffenhofen, Germany, November 2018.

[21] Robert Pritchett, Kathleen C Howell, and David C. Folta. Low-thrust trajectory design for a cislunar cubesat leveraging structures from the bicircular restricted four-body problem. In *70th International Astronautical Congress*, pages 1–18, Washington D.C., USA, October 2019.

[22] Navid Nakhjiri and Benjamin Villac. Automated stable region generation, detection, and representation for applications to mission design. *Celestial Mechanics and Dynamical Astronomy*, 123(1):63–83, 2015.

[23] Benjamin Villac, Rodney Anderson, and Alex Pini. Computer aided ballistic orbit classification around small bodies. *The Journal of the Astronautical Sciences*, 63(3):175–205, 2016.

[24] Natasha Bosanac. Applications of clustering to higher-dimensional poincaré maps in multi-body systems. In *29th AAS/AIAA Space Flight Mechanics Meeting*, pages 1–14, Ka'anapali, Hawaii, January 2019. American Astronautical Society.

[25] Stefano Bonasera and Natasha Bosanac. A data mining approach to using poincaré maps in multi-body trajectory design strategies. In *20th AIAA Scitech Forum*, pages 1–20, Orlando, Florida, January 2020. AIAA.

[26] Yu Gao, Tianshe Yang, Minqiang Xu, and Nan Xing. An unsupervised anomaly detection approach for spacecraft based on normal behavior clustering. In *2012 Fifth International Conference on Intelligent Computation Technology and Automation*, pages 478–481. IEEE, 2012.

[27] Jacob Broida and Richard Linares. Spacecraft rendezvous guidance in cluttered environments via reinforcement learning. In *29th AAS/AIAA Space Flight Mechanics Meeting*, pages 1–15, Ka'anapali, Hawaii, January 2019. American Astronautical Society.

[28] Brian Gaudet, Roberto Furfaro, and Richard Linares. Reinforcement learning for angle-only intercept guidance of maneuvering targets. *Aerospace Science and Technology*, 99, 2020.

[29] Davide Guzzetti. Reinforcement learning and topology of orbit manifolds for station-keeping of unstable symmetric periodic orbits. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–20, Portland, Maine, August 2019. American Astronautical Society.

[30] Jason A. Reiter, David B. Spencer, and Richard Linares. Spacecraft maneuver strategy optimization for detection avoidance using reinforcement learning. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–19, Portland, Maine, August 2019. American Astronautical Society.

[31] Jason A. Reiter and David B. Spencer. Augmenting spacecraft maneuver strategy optimization for detection avoidance with competitive coevolution. In *20th AIAA Scitech Forum*, Orlando, Florida, January 2020. AIAA.

[32] Ashwati Das-Stuart, Kathleen C Howell, and David C. Folta. A rapid trajectory design strategy for complex environments leveraging attainable regions and low-thrust capabilities. In *68th International Astronautical Congress*, pages 1–19, Adelaide, Australia, September 2017.

[33] Ashwati Das-Stuart and Kathleen Howell. Contingency planning in complex dynamical environments via heuristically accelerated reinforcement learning. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–21, Portland, Maine, August 2019. American Astronautical Society.

[34] Daniel Miller and Richard Linares. Low-thrust optimal control via reinforcement learning. In *29th AAS/AIAA Space Flight Mechanics Meeting*, pages 1–18, Ka'anapali, Hawaii, January 2019. American Astronautical Society.

[35] Daniel Miller, Jacob A. Englander, and Richard Linares. Interplanetary low-thrust design using proximal policy optimization. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–16, Portland, Maine, August 2019. American Astronautical Society.

[36] Roberto Furfaro, Andrea Scorsoglio, Richard Linares, and Mauro Massari. Adaptive generalized zem-zev feedback guidance for planetary landing via a deep reinforcement learning approach. *Acta Astronautica*, 171:156–171, 2020.

[37] Brian Gaudet and Richard Linares. Integrated guidance and control for pinpoint mars landing using reinforcement learning. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–20, Snowbird, Utah, August 2018. American Astronautical Society.

[38] Brian Gaudet, Richard Linares, and Roberto Furfaro. Deep reinforcement learning for six degree-of-freedom planetary landing. *Advances in Space Research*, 65(7):1723–1741, 2020.

[39] Andrea Scorsoglio, Roberto Furfaro, Richard Linares, and Brian Gaudet. Image-based deep reinforcement learning for autonomous lunar landing. In *20th AIAA Scitech Forum*, Orlando, Florida, January 2020. AIAA.

[40] Brian Gaudet, Richard Linares, and Roberto Furfaro. Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations. *Acta Astronautica*, 171:1–13, 2020.

[41] Brian Gaudet, Richard Linares, and Roberto Furfaro. Six degree-of-freedom hovering over an asteroid with unknown environmental dynamics via reinforcement learning. In *20th AIAA Scitech Forum*, Orlando, Florida, January 2020. AIAA.

[42] Christopher J. Sullivan and Natasha Bosanac. Using reinforcement learning to design a low-thrust approach into a periodic orbit in a multi-body system. In *20th AIAA Scitech Forum*, Orlando, Florida, January 2020. AIAA.

[43] Nicholas B. LaFarge, Daniel Miller, Kathleen C. Howell, and Richard Linares. Guidance for closed-loop transfers using reinforcement learning with application to libration point orbits. In *20th AIAA Scitech Forum*, Orlando, Florida, January 2020. AIAA.

[44] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer, 2nd ed. edition, 2009.

[45] Shane Robinson, Sara Scarritt, and John L. Goodman. Encke-beta predictor for orion burn targeting and guidance. In *39th Annual AAS Guidance and Control Conference*, pages 709–731, Breckenridge, CO, February 2016. AAS.

[46] Sanjeevi Sirisha Karri. Nasa sbir 2019 phase i solicitation: Deep neural net and neuromorphic processors for in-space autonomy and cognition. online, 2019.

[47] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. A survey of neuromorphic computing and neural networks in hardware. *CoRR*, abs/1705.06963, 2017.

[48] Gennadi Bersuker, Maribeth Mason, and Karen L. Jones. Neuromorphic computing: The potential for high-performance processing in space. Technical report, The Aerospace Corporation, 2018.

[49] Peter Doggers. Lc0 wins computer chess championship, makes history. Chess.com, Apr 2020.

[50] Peter Doggers. Leela chess zero beats stockfish 106-94 in 13th chess.com computer chess championship. Chess.com, Apr 2020.

[51] Leela zero. https://github.com/leela-zero/leela-zero, Apr 2020.

[52] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[53] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518, Feb 2015.

[54] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[55] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229 –256, May 1992.

[56] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[57] Marc D Rayman, Philip Varghese, David H Lehman, and Leslie L Livesay. Results from the deep space 1 technology validation mission. *Acta Astronautica*, 47(2):475–487, 2000.

[58] Christopher Russell and Carol Raymond. *The Dawn Mission to Minor Planets 4 Vesta and 1 Ceres*. Springer, 1st ed. edition, 2012.

[59] David Irimies, David Manzella, and Timothy Ferlin. Summary of gateway power and propulsion element (ppe) studies. In *2019 IEEE Aerospace Conference*, volume 2019-, pages 1–6, Big Sky, Montana, 2019. IEEE.

[60] Andrew Cox, Kathleen Howell, and David Folta. Dynamical structures in a low-thrust, multi-body model with applications to trajectory design. *Celestial Mechanics and Dynamical Astronomy*, 131(3):1–34, 2019.

[61] Hitoshi Kuninaka, Kazutaka Nishiyama, Yukio Shimizuand Ikko Funaki, and Hiroyuki Koizumi. Hayabusa asteroid explorer powered by ion engines on the way to earth. In *31st International Electric Propulsion Conference*, pages 1–6, Ann Arbor, Michigan, September 2009.

[62] Kazutaka Nishiyama, Satoshi Hosoda, Kazuma Ueno, Ryudo Tsukizaki, and Hitoshi Kuninaka. Development and testing of the hayabusa2 ion engine system. *Transactions of the Japan Society for Aeronautical and Space Sciences*, 14(30):131–140, July 2016.

[63] Natasha Bosanac, Andrew D Cox, Kathleen C Howell, and David C Folta. Trajectory design for a cislunar cubesat leveraging dynamical systems techniques: The lunar icecube mission. *Acta Astronautica*, 144:283–296, 2018.

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

[65] Davide Guzzetti, Emily M. Zimovan, Kathleen C. Howell, and Diane C. Davis. Stationkeeping analysis for spacecraft in lunar near rectilinear halo orbits. In *27th AAS/AIAA Space Flight Mechanics Meeting*, pages 1–24, San Antonio, Texas, February 2017. American Astronautical Society.

[66] Collin E. York and Kathleen C. Howell. A two-level differential corrections algorithm for low-thrust spacecraft trajectory targeting. In *29th AAS/AIAA Space Flight Mechanics Meeting*, pages 1–20, Ka'anapali, Hawaii, January 2019. American Astronautical Society.

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

[68] George Bozis. Zero velocity surfaces for the general planar three-body problem. *Astrophysics and Space Science*, 43(2):355–368, 1976.

[69] Diane Davis, Sagar Bhatt, Kathleen Howell, Jiann-Woei Jang, Ryan Whitley, Fred Clark, Davide Guzzetti, Emily Zimovan, and Gregg Barton. Orbit maintenance and navigation of human spacecraft at cislunar near rectilinear halo orbits. In *27th AAS/AIAA Space Flight Mechanics Meeting*, San Antonio, Texas, February 2017. American Astronautical Society.

[70] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirthy-Second AAAI Conference On Artificial Intelligence (AAAI)*, New Orleans, Louisiana, February 2018.

[71] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, July 2018.

[72] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

A. REINFORCEMENT LEARNING PARAMETER SELECTION

Parameter selection and tuning is an important aspect in both PPO learning and RL environment design. Well-performing agents may be produced with different combinations of parameters. Values employed in this research are summarized in Table A.1, with additional suggested values included to indicate cases where desirable behavior is observed given different parameter values. Furthermore, the specific configurations of the employed neural networks are listed in Table A.2. These networks are implemented using TensorFlow [72].

Table A.1. Suggested parameter values for PPO training and CR3BP RL environment configuration.

Variable name	Symbol	Value
Discount factor	γ	0.88
Number of optimization epochs (actor)		20/batch
Number of optimization epochs (critic)		10/batch
Batch frequency		20 episodes
Reward steepness	λ	340
Reward scaling gradient	ξ	1
Reward divergence penalty	p	-4
Actor learning rate		0.00011
Critic learning rate		0.00204
KL Divergence Target	d_{targ}	0.003
Number of Training Episodes		120,000

Table A.2. Configuration of actor and critic neural networks employed in this investigation.

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