Low-thrust Spacecraft Guidance and Control using Proximal Policy Optimization

by

Daniel Miller

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2020

© Massachusetts Institute of Technology 2020. All rights reserved.

Author .....
Department of Aeronautics and Astronautics
May 19, 2020

Certified by .....
Richard Linares
Charles Stark Draper Assistant Professor
Thesis Supervisor

Accepted by .....
Sertac Karaman
Associate Professor of Aeronautics and Astronautics

Low-thrust Spacecraft Guidance and Control using Proximal Policy Optimization

by

Daniel Miller

Submitted to the Department of Aeronautics and Astronautics
on May 19, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Artificial intelligence is a rapidly developing field that promises to revolutionize spaceflight with greater robotic autonomy and innovative decision making. However, it remains to be determined which applications are best addressed using this new technology. In the coming decades, future spacecraft will be required to possess autonomous guidance and control in the complex, nonlinear dynamical regimes of cis-lunar space. In the realm of trajectory design, current methods struggle with local minima, and searching large solutions spaces. This thesis investigates the use of the Reinforcement Learning (RL) algorithm Proximal Policy Optimization (PPO) for solving low-thrust spacecraft guidance and control problems. First, an agent is trained to complete a 302 day mass-optimal low-thrust transfer between the Earth and Mars. This is accomplished while only providing the agent with information regarding its own state and that of Mars. By comparing these results to those generated by the Evolutionary Mission Trajectory Generator (EMTG), the optimality of the trajectory designed using PPO is assessed. Next, an agent is trained as an onboard regulator capable of correcting state errors and following pre-calculated transfers between libration point orbits. The feasibility of this method is examined by evaluating the agent's ability to correct varying levels of initial state error via Monte Carlo testing. The generalizability of the agent's control solution is appraised on three similar transfers of increasing difficulty not seen during the training process. The results show both the promise of the proposed PPO methodology and its limitations, which are discussed.

Thesis Supervisor: Richard Linares
Title: Charles Stark Draper Assistant Professor

Acknowledgments

Completing this thesis would have been impossible without the help of so many others. First, I would like thank my advisor, Prof. Richard Linares. Without his intellectual brilliance, I would never have been able to devise the research project contained within these pages. Without his investment of time into my own development as a researcher, following through on that idea would have been equally impossible.

Next, I would like to thank Dr. Jacob Englander and the entire Code 595 team at Goddard Space Flight Center. Your insights into the world of mission planning and trajectory optimization have been illuminating. I would also like to thank the combined faculties of the Department of Aeronautics and Astronautics at the Massachusetts Institute of Technology, the Department of Aerospace Engineering and Mechanics at the Universty of Minnesota, and the Department of Mechanical and Aerospace Engineering at the State University of New York at Buffalo. Your valuable instruction has been critical in my growth as an engineer.

Arriving at this point would have been impossible without the assistance of mentors outside of academia and the professional sphere. I would like to thank Mr. John Schwartz and Dr. Ken Pickover for their encouragement of my interest in spaceflight. I would also like to thank Dr. Paul Gilmore, Dr. John Sorkin, and Dr. Albert Medina for planting the idea into my mind that maybe I should go to grad school after all.

I would like to take a moment to direct my attention to my friends and labmates: Thank you for listening patiently to all of my melodramatic ramblings about reinforcement learning and the stressors of graduate education over the last 3 years. A thesis may be a solo activity, but graduate school is truly a team effort. Finally, I would like to thank my true advisors: my parents. I would never have been able to arrive at this point in my life without your love and guidance.

This work was supported by NASA Space Technology Research Fellowship. I would also like to acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC and consultation resources that have contributed to the research results reported within this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the coming decades, the planning and execution of exploration missions will dramatically change as the confluence of new technologies unlock new orbits and spacecraft capabilities. Low-thrust propulsion has matured and has been utilized on deep space missions such as BepiColombo, Dawn, Hayabusa, and Hayabusa2. Perhaps most consequentially, it has been selected for the Lunar Gateway project, which will use a Solar Electric Propulsion (SEP) system three times more powerful than those currently available [33]. Due to the far greater specific impulse of SEP compared to chemical propulsion, the proportion of spacecraft mass dedicated to fuel can be greatly reduced, thus freeing it for other purposes [47]. For the proposed return to the Moon with the Artemis missions and the eventual Lunar Gateway, NASA intends to utilize Near Rectilinear Halo Orbits (NRHO). Such an orbit would offer easy access from Earth, constant communication with Mission Control, low station-keeping fuel expenditures, and a favorable thermal environment. However, unlike the Apollo missions, this will require the leveraging of multi-body orbital mechanics [72]. Furthermore, the Orion spacecraft to be used on these missions must be capable of providing a large degree of autonomy in all phases of flight, including automatic rendezvous and docking, ascent aborts, and deorbit burns [32]. The ability to generate trajectories that can leverage multi-body dynamics and low-thrust propulsion will be a critical capability in the future, as will the autonomous capabilities necessary to execute them. Current technology leaves significant room for improvement in these

areas.

1.1 Literature Review

1.1.1 Current Trajectory Design Techniques

Traditional optimization for trajectory design is a complex process requiring the consideration of many mission parameters. A designer must consider flight time, launch windows, and the science targets of an intended mission. Interplanetary missions may also require the selection of an appropriate sequence of flybys. Spacecraft specifications such as mass, thrust, and power generation must be factored in. Low-thrust and multi-body dynamics add additional complications. While missions utilizing chemical propulsion can approximate maneuvers as instantaneous changes in velocity, low-thrust mission planners can only make similar simplifications early in the trajectory design process. To achieve a highly accurate solution, any such simplification must eventually be eliminated due to the days or weeks over which maneuvers occur. Instead, a continuous control trajectory over the course of the mission must be created. While two-body orbital mechanics have an analytical solution, the addition of further bodies make such a solution impossible [55]. A patched-conics solution may be used up to a point, but eventually a numerical solution becomes necessary. Despite these challenges, optimization methods for trajectory and mission design have been very successful. While a complete history and review of these methods is beyond the scope of this thesis, a broad and simplified overview is provided here for motivation.

At the highest level, these techniques can broadly be split into two categories: direct and indirect methods. Indirect methods are based on the calculus of variations and pose optimization tasks in the form of one or more two-point boundary value problems. They can rapidly converge to a solution, but suffer from a small convergence radius, thus necessitating a good initial guess. Indirect optimization also often requires analytic derivatives of the objective function with respect to the state, which may not be available [9], and are sensitive to multiplier terms in the objective

function known either as costates or adjoints. Direct methods, in contrast, pose a given optimal control problem as a nonlinear programming problem (NLP). They are more robust and can more easily accept path constraints, at the cost of increased computational expense [68]. Most low-thrust trajectory design tools fall into this latter category, which shall be the focus of the remainder of this discussion.

Direct methods can be further divided based on the method by which decision variables (e.g. control inputs) and boundary conditions are transcribed. The first group are shooting methods. The basic single shooting approach is to define a set of initial conditions, end conditions, and other parameters that are within the objective function or constraints. The optimizer then solves an initial value problem that attempts to find a control input that will, starting from the given initial condition, satisfy the end conditions, minimize the objective function, and satisfy constraints. If the problem contains inequality constraints or discrete changes in the problem such as the jettisoning of a rocket stage, the problem is often divided into subproblems known as phases [9].

One difficulty with single shooting is that by solving the optimization problem all at once, small changes early in the trajectory can lead to large errors later when trying to meet the end conditions. Multiple shooting addresses this issue by dividing the problem into a series of smaller segments. Each segment is then individually solved as a single shooting initial value problem. To ensure a continuous solution, constraints are defined that require the end condition of one segment to match the initial condition of the subsequent segment [9]. The second group are known as direct collocation methods. These methods begin with an arbitrary polynomial that is a function of time and is composed of the sum of terms weighted by coefficients. The user must also provide an initial guess of the state and control trajectory. The purpose of this polynomial is to approximates the state variables, with its derivative therefore approximating the equations of motion. The currently unknown trajectory is divided into segments in time, with the points at the beginning and end of each segment known as a node and defined by a state and control input. The coefficients of the polynomial are then calculated to match the state at each node. Boundary conditions

are thus satisfied.

In the interior of each segment is what is known as a collocation point. The solver calculates the difference between the approximated state derivative and the equations of motion at these points. To optimize the trajectory, the solver attempts to minimize this differential error by adjusting the states and control inputs at the nodes. The process then repeats by resolving the polynomial's coefficients. In short, the polynomial satisfies the state variables of the trajectory at the nodes and the derivatives of the variables at the collocation points. Many methods exist for the placement of nodes and collocation points, such as Hermite-Simpson, Gauss-Lobato, and pseudospectral [68], but this is well outside the scope of this review.

During the mission design process, NASA engineers utilize a wide range of tools depending on their center affiliation and the level of fidelity required. According to Beeson et al. [7], programs such as the Jet Propulsion Laboratory's (JPL) Satellite Tour Design Program – Low Thrust, Gravity Assist (STOUR-LTGA) may be used at the lowest level of fidelity to conduct a grid search for feasible trajectories. However, this models a trajectory using geometric arcs and is not an optimization tool at all [48]. Its solution can be used as an initial guess for medium fidelity tools such as JPL's Mission Analysis Low-Thrust Optimization (MALTO) [63] or Gravity Assist Low-Thrust Local Optimization Program (GALLOP) [39]. Both use direct transcription (shooting) using the Sims-Flanagan method [7], which models the thrust of a spacecraft's continuous propulsion system as a series of impulsive maneuvers [64]. Programs at this level will also frequently use patched-conic orbital mechanics [20], which will be unable to simulate the multi-body dynamics necessary for the cis-lunar operations envisaged for the Artemis program.

For the most detailed and accurate trajectories, tools such as the General Mission Analysis Toolkit (GMAT), Copernicus, the Mission Analysis, Operations, and Navigation Toolkit Environment (MONTE), or AGI's Systems Tool Kit (STK) are used. These use the full equations of motion without simplification, account for solar radiation pressure, and include the simultaneous gravitational attraction of many major and minor bodies within the solar system. Interestingly, Copernicus can solve

either direct or indirect optimization problems [45]. Also, due to the lack of simplifications applied to the dynamics, high-fidelity optimizers such as Copernicus are capable of handling multi-body dynamics to libration point orbits as well as interplanetary missions [74].

In contrast to their colleagues at JPL, researchers at Goddard Space Flight Center (GSFC) perform both medium and high-fidelity low-thrust trajectory design using a single, modular, scalable-fidelity optimization tool known as Evolutionary Mission Trajectory Generator (EMTG) [20]. This program is capable of both low fidelity global search and trade studies and high-fidelity solution optimization. EMTG can also be paired with global search heuristics in order to avoid local minima, but such a strategy can be computationally expensive [21, 18, 19].

While effective, this approach to mission design and trajectory optimization is consuming in both time and computational expense. Before progressing to the high-fidelity optimizers, a high-quality medium-fidelity solution must be produced to use as an initial guess. For challenging problems, this places a time-consuming human-in-the-loop process in the trajectory design workflow [20]. During the initial stages of trajectory planning, the low-fidelity search of the solution space requires examining thousands of possible solutions [16]. When solving problems with large numbers of parameters using higher-fidelity direct methods, the solution space may also become so large as to make the problem seemingly intractable, a phenomenon commonly referred to as the curse of dimensionality. There is also the more general cost of solving a trajectory multiple times at different levels of fidelity. As an additional complication, lower-fidelity methods may be unable to work with multi-body dynamics, necessitating the use of higher fidelity methods.

Neither a direct nor indirect optimization method, dynamic programming exists as an alternative approach to optimization trajectory problems. One variant, Differential Dynamic Programming (DDP), has been researched in recent years for low-thrust trajectory problems. In 2017, Aziz et al. used DDP to design trajectories for an SEP spacecraft while accounting for discontinuities in the power available to the spacecraft due to solar eclipses [4]. In 2018, Aziz, Scheeres, and Lantoine optimized

transfers between distant retrograde orbits (DROs), planar lyapunov orbits, and Halo orbits, all within the the CR3BP model of cis-lunar space [6]. By using a Sundman transformation to change the independent variable of motion from time to the true, mean, or eccentric anomaly, Aziz et al. optimized low-thrust orbit raising maneuvers composed of up to 2000 revolutions. To accomplish this, supercomputing resources were required [5].

1.1.2 The Need for Spacecraft Autonomy

Due to the extended nature of its missions and the potentially small crew to operate it, the Orion spacecraft of the upcoming Artemis missions will provide substantially greater automated guidance, navigation, and control (GN&C) capabilities compared to the previous Space Shuttle program. In addition to the commonly automated launch sequence, rendezvous, proximity operations, docking, and deobrit maneuvers will all be controlled by the onboard computer. It is also required to be capable of returning the crew to Earth without ground communications [32]. All of this will be accomplished with the limited computational power available on a radiation hardened version of the decades-old PowerPC 750 CPU [49]. As a result, low computational overhead is a desirable trait for any onboard application.

Unfortunately, the mission planning tools discussed previously do not fit this requirement, often requiring many hours of run time using supercomputer resources [21, 69]. While the design of entire mission trajectories is not a requirement for flight computers either on Orion or any robotic mission, it is clear that doing so would be infeasible using methods that are dependent on abundant computational resources. For future missions to Mars and beyond, the concerns that mandated the increase in autonomy for the Orion capsule can only be expected to become more intense. This suggests that the development of new methods of trajectory optimization capable of operating on space-rated hardware will eventually be necessary. At the very least, new technologies capable of onboard guidance and control, even if not used for planning entire trajectories, would be a beneficial development.

1.1.3 Machine Learning in Astrodynamics

Within the broad category of machine learning, there are two methods of training a computer as a controller and optimizer for spacecraft GN&C applications: supervised learning and reinforcement learning. In supervised learning, researchers provide the computer with a curated set of training data. This is often used in regression or classification tasks and is an effective way of mapping known inputs to outputs. In contrast, Reinforcement Learning (RL) instructs the computer by providing feedback to repeated attempts at solving a given problem [66]. Most modern algorithms of either learning arrangement use artificial neural networks to approximate nonlinear functions to map inputs to outputs that decide on actions to take. Deep RL, which uses multi-layer artificial neural networks, made headlines in the mid-2010s, in part due to the groundbreaking work of the Google Deep Mind. In 2015, Mnih et al. demonstrated the use of the deep Q-learning algorithm to train computer agents capable of playing 49 different classic Atari games, 29 of which were at a human or superior skill level [43]. The following year, Silver et al. trained an agent capable of beating the European champion of the Chinese game of Go in a series of matches 5 to 0. Mastering this complex game of an estimated 4.9×10^{359} possible combinations of moves was achieved using a mixture of supervised learning and reinforcement learning. [61]. This was followed by a further development in which the agent demonstrated super-human performance via pure reinforcement learning [62] and testifies to RL's ability to learn novel solutions – a key advantage over supervised learning. These developments demonstrated the potential of RL solutions to other seemingly intractable problems that suffer from the curse of dimensionality. As an additional bonus, while significant computational power was utilized to train these agents, neural networks themselves are low-cost to operate once trained, requiring only matrix multiplication, addition, and the element-wise application of a scalar nonlinear function.

While the use of neural networks in spacecraft GN&C research is not a new one – Dachwald used a shallow neural network to optimize low-thrust interplanetary trajectories in 2004 [12] – recent advancements in machine learning algorithms and the use

of deep, or multi-layer, artificial neural networks have made them a more effective tool for researchers. In 2018, Parrish and Scheeres used a deep neural network to make corrections to perturbed low-thrust trajectories by training the agent on pre-optimized trajectories in the vicinity of the nominal path [46]. DeSmet and Scheeres used an artificial neural network to identify heteroclinic connections between L_1 and L_2 Lyapunov orbits [65]. Furfaro et al. used supervised learning to create a controller capable of predicting the fuel-optimal thrust direction and magnitude for lunar landing based solely on images taken by the simulated spacecraft [23].

Das-Stuart, Howell, and Folta created a unique combined approach of supervised learning, reinforcement learning, and traditional optimization methods for low-thrust path planning in cis-lunar space. Based on the performance capabilities of the spacecraft, regions of space accessible to the spacecraft were identified. Path planning was then accomplished using reinforcement learning [14]. Later work further developed this approach by identifying potential waypoints for the path based on supervised learning and then learning to connect them into a complete path using reinforcement learning. [13] In both cases, the final path was further optimized using traditional methods. Das-Stuart and Howell later applied these methods to plan trajectories to recover missions after path deviations occur due to temporarily compromised thrusting capabilities [15].

Proximal Policy Optimization (PPO) has proven to be a popular and effective RL algorithm for the creation of closed-loop optimal controllers for space-based problems. Gaudet, Linares, and Furfaro have published extensively on multiple problems using this algorithm, including six-degree of freedom hovering and proximity operations around asteroids [26, 27, 25] and six-degree of freedom Mars landing [24]. Scorsoglio, Furfaro, Linares, and Gaudet further developed Furfaro et al.'s supervised learning approach from [23] into a pure-RL approach to lunar landing based on image and altimeter inputs [59]. Reiter, Spencer, and Linares used PPO and an adverserial training algorithm to train an agent capable of avoiding on-orbit detection by ground-based sensors [51, 52].

1.2 Impact and Outline

Although current trajectory optimization methods are effective given unlimited time and computational resources, they are poorly suited for onboard applications and struggle with avoiding local minima. With its ability to find unique solutions to seemingly intractable problems with large solution spaces and the low-cost of neural network closed-loop controllers, RL has the ability to solve both of these issues. Therefore, the author seeks to investigate the follow two questions. The first is to determine the capabilities of an RL optimizer for low-thrust mission planning compared to a traditional direct method program. The second is whether an RL agent is suitable for onboard applications in situations featuring complex dynamics. The first scenario will focus primarily on the optimality of the solution, while the latter places a greater emphasis on the controller's ability to resolve different possible on-orbit scenarios. This is because feasibility is more important than optimality for an onboard GN&C system [38].

The author has used PPO to develop the methods presented in this thesis over the last 3 years. Miller and Linares used PPO to train an agent and solve a low-thrust transfer between two Distant Retrograde Orbits (DROs) in the planar CR3BP model without any pre-generated optimal trajectory information [42]. Miller, Englander, and Linares then used the same method to plan a mass-optimal low-thrust transfer between the Earth and Mars in three-dimensions [41]. Most recently, LaFarge, Miller, Howell, and Linares presented a closed-loop controller trained using PPO capable of correcting for perturbations and completing a low-thrust transfer between planar Lyapunov orbits. The same agent could generalize this knowledge and successfully complete the same task on other, previously unseen transfers [36].

The remainder of the document will be organized as follows. In Chapter 2, a complete explanation of the necessary machine learning theory will be provided. This will include an explanation of the operation and optimization of artificial neural networks, a history and derivation of gradient descent methods, and the underlying theory of RL and PPO. In Chapter 3, an agent will be trained to complete an Earth-Mars transfer

and compared against a corresponding solution provided by EMTG. In Chapter 4, an agent will be trained to correct for position and velocity perturbations and complete low-thrust transfers in cis-lunar space. Monte Carlo testing will be used to measure the generalizability of the closed-loop controller. Finally, Chapter 5 will summarize these findings and provide an answer to the two aforementioned questions.

Chapter 2

Reinforcement Learning

In this chapter, Proximal Policy Optimization will be explained from first principles. To begin, the operation of neural networks and their training via backpropagation will be discussed. This will be followed by sections regarding Adam, a gradient descent method used to optimize neural networks, the basics of reinforcement learning, and the actor-critic framework used by PPO. Finally, Proximal Policy Optimization is explained in detail starting from the broader category of Policy Optimization methods. A sense of the history and evolution of these methods has been included in order to provide the reader with an understanding of not just how these methods work, but also why their individual features are necessary. It is hoped that this will provide a deeper comprehension of the theory that will both aid in understanding this work and assist others in their future research.

2.1 Neural Networks

Artificial neural networks (ANNs) are one of the most important tools that an RL researcher has. They provide a means for complex, non-linear functions to be modelled purely based on input and output data without any knowledge of the form of the function itself. The early history of ANNs begins in the 1940s [40] and their use as a function approximator for RL originates in the 1950s [22]. However, it is only in the last decade that deep, or multi-layer, ANNs have become the powerful mechanism

for machine learning that they are so well known for today [8, 28, 66].

Section 2.1.1 will discuss how to use a neural network to produce an output given an input. Section 2.1.2 explains how to then optimize the parameters of a neural network in order to produce a known, correct output for a given input. These first two sections rely heavily on the work of Michael Nielsen [44]. Finally, Section 2.1.3 will explain how gradient descent methods evolved into Adam optimizer, the gradient descent method used in this thesis to train a neural network-based agent.

2.1.1 Forward Propagation and Execution

Neural networks approximate a nonlinear function through a series of matrix operations and element-wise nonlinear functions. A neural network is composed of layers. For a layer l, there exists a weights matrix w^l, a bias matrix b^l, and a nonlinear function σ. For a given input, o^{l-1}, the layer produces an output o^l via the following linear algebra operation.

$$\begin{aligned} p^l &= w^l o^{l-1} + b^l \\ o^l &= \sigma\left(p^l\right) \end{aligned} \tag{2.1}$$

While its importance will only later be made clear, Eq. 2.1 defines p^l as the output of the matrix operations portion before the element-wise nonlinear function is applied. Common selections for these functions, usually referred to as activation functions, include sigmoid, tanh, and rectified linear unit (ReLU). Without them, it would be impossible for a NN to model a nonlinear function because p^l is produced via purely linear operations.

The neuron is a word commonly associated with neural networks. These are the sub-components of each layer. Consider the generic NN shown in Figure 2-1. In Layer 2, there are five neurons with each neuron containing a single value. Looking back to Eq. 2.1, this means that p^2 is a 5×1 column vector that was produced by multiplying a 5×3 weights matrix w^2 with a 3×1 Layer 1 output o^1 and adding a 5×1 b^2 vector.

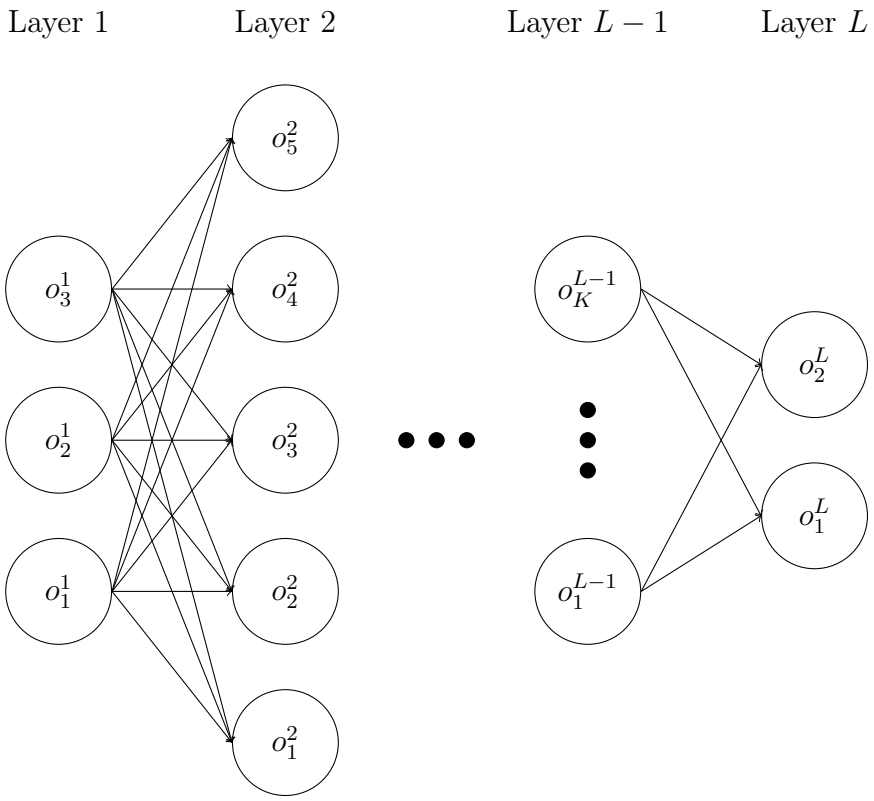

Figure 2-1: A Neural Network

The activation function is then applied element-wise to create the 5×1 o^2.

In the next section, it will be necessary to look at these equations at the scale of each neuron, rather than the layer itself. Eq. 2.1 can be expressed in this manner through the introduction of a new subscript notation. Variables that previously referred to a vector of neurons – o^l, p^l, and b^l – all receive a single subscript indicating the specific neuron in the layer to which they refer. The weighting matrix receives two subscripts to denote the link between two specific neurons in two different layers. Using this notation, weight w_{jk}^l is the individual weight in the weighting matrix w^l that links neuron output o_k^{l-1} to p_j^l. In the ANN diagram, the weights are represented by the web of arrows that link neurons between layers.

Algorithm 1 Feedforward NN Execution

procedure FEEDFORWARD(ξ, A neural network)
  $o^1 \leftarrow \xi$ ▷ Set the input
  for $l = 2, 3, \ldots, L$ **do**
    $p^l \leftarrow w^l o^{l-1} + b^l$
    $o^l \leftarrow \delta_j^l$
  end for
end procedure

$$\begin{aligned} p_j^l &= \left(\sum_k w_{jk}^l o_k^{l-1} + b_j^l \right) \\ o_j^l &= \sigma \left(\sum_k w_{jk}^l o_k^{l-1} + b_j^l \right) \end{aligned} \tag{2.2}$$

Eq. 2.2 shows that the neuron output o_j in layer l is equal to the activation function applied to the sum of the individual matrix operations that link it to the neurons of the previous layer. In the representation of a NN shown in Fig. 2-1, this is equivalent to the sum of the arrows that enter a given neuron with the activation function applied.

2.1.2 Backpropagation

In order to optimize a neural network, a process known as a backpropagation is used to calculate the necessary gradient for each and every weight and bias. First, an objective function is defined. For the purposes of this explanation, assume that it is quadratic, where ψ is the correct NN output that is desired given the input ξ.

$$J(\xi) = \frac{1}{2n} \sum_{\xi} \left\| \psi - o^L(\xi) \right\|^2 \tag{2.3}$$

Additionally, assume that the objective function for an entire data set of inputs ξ and desired outputs ψ is the average of the objective function applied to each individual pair of data. With n such input-output pairs,

$$J_\xi = \frac{1}{2}\left\|\psi - o^L(\xi)\right\|^2$$
$$J = \frac{1}{n}\sum_\xi J_\xi \tag{2.4}$$

this will allow for partial derivatives to later be calculated for the objective function as applied to single training examples J_ξ and for the partial derivative over the entire dataset to be calculated by averaging the individual partials. Additionally, since each output is paired to a specific input, assume that the function can be written purely as a function of the input.

Since the goal of optimizing the neural network is to find the weights and biases that minimize the objective function, the change in cost function with respect to a change in the ANN is calculated. This is defined as the error δ_j^l of the jth neuron in the lth layer. Due to the quadratic nature of Eq. 2.3, the objective function is concave up. The global minimum will thus be located where the derivative of the objective function is zero.

$$\delta_j^l \equiv \frac{\partial J}{\partial p_j^l} \tag{2.5}$$

Much as forward propagation demonstrated how an input works its way through a neural network, backpropagation starts at the output and moves back up the network. Therefore, it is necessary to start by looking at the output layer. From Eq. 2.4, the activation in a given neuron in the final layer is $o_j^L = \sigma\left(p_j^L\right)$. By taking the derivative, the change in the output of the jth neuron with respect to the pre-activation value is found.

$$\frac{\partial o_j^L}{\partial p_j^L} = \sigma'\left(p_j^L\right) \tag{2.6}$$

Using Eqs. 2.5 and 2.6, the error at at the final layer can be written as the product of the change in the objective function with respect to the output of the neural network, $\frac{\partial J}{\partial o_j^L}$, and the rate of change of objective function, $\sigma'\left(p_j^L\right)$.

$$\delta_j^L = \frac{\partial J}{\partial o_j^L} \sigma' \left(p_j^L \right) \tag{2.7}$$

To provide some insight into the deeper meaning of this equation, if the first term below is small, then the function to be minimized is not very sensitive to the jth neuron's output. The second term is a measure of the speed with which the activation function is changing at p_j^L.

This is the first of four equations that are critical to backpropagation. However, it is often more useful to write this equation in a vectorized form. This requires the use of a Hadamard Product, $\odot$, which is an element-wise multiplication operator. Eq. 2.7 now becomes

$$\delta^L = \nabla_o J \odot \sigma' \left(p^L \right) \tag{2.8}$$

The second equation is used to calculate the error at the previous layer of the network based on the layer ahead of it – that is, to find the error δ^l based upon δ^{l+1}. This calculation calculation can then be repeated started from the output layer and propagated backwards towards the input layer for a network of an arbitrary size. The proof needed to calculate this is simplest in the neuron notation. Consider the error in neuron j of layer l as given in Eq. 2.14.

$$\begin{aligned} \delta_j^l &= \frac{\partial J}{\partial p_j^l} \\ &= \sum_k \frac{\partial J}{\partial p_k^{l+1}} \frac{\partial p_k^{l+1}}{\partial p_j^l} \end{aligned} \tag{2.9}$$

The first term can be recognized as Eq. 2.14. Additionally, as scalar terms, the commutative property applies and the order can be swapped.

$$\delta_j^l = \sum_k \frac{\partial p_k^{l+1}}{\partial p_j^l} \delta_k^{l+1} \tag{2.10}$$

While the error of a neuron in layer l is now shown as a function of the error in

layer $l+1$, the first term above is not especially helpful. Fortunately, it can easily be replaced by something more familiar. First, Eq. 2.2 for layer $l+1$ is written in terms of layer l.

$$\begin{aligned} p_j^{l+1} &= \sum_k w_{jk}^{l+1} o_k^l + b_j^{l+1} \\ &= \sum_k w_{jk}^{l+1} \sigma\left(p_k^l\right) + b_j^{l+1} \end{aligned} \tag{2.11}$$

By differentiating this equation, $\frac{\partial p_k^{l+1}}{\partial p_j^l}$ can be found.

$$\frac{\partial p_k^{l+1}}{\partial p_j^l} = w_{kj}^{l+1} \sigma'\left(p_j^l\right) \tag{2.12}$$

Substituting this into Eq. 2.10 and rearranging the scalar terms, we obtain the desired error equation.

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'\left(p_j^l\right) \tag{2.13}$$

As discussed with Eqs. 2.1 and 2.2, the matrix version can be recovered using the summation. Therefore, the desired matrix equation is as follows, with the transpose added to ensure the correct matrix dimensions.

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(p^l\right) \tag{2.14}$$

Two more equations are necessary for backpropagation. The third equation shows that the rate of change of the objective function with respect to the bias of any neuron is the error of that neuron itself. To show this, first take the derivative of p_j^l from Eq. 2.2 with respect to bias b_j^l. Notice that the summation in the equation is for neuron k, so the bias term is on the outside of the summation.

$$\frac{\partial p_j^l}{\partial b_j^l} = 1 \tag{2.15}$$

A quick application of the chain rule then yields the desired equation.

$$\begin{aligned}\delta_j^l &= \frac{\partial J}{\partial p_j^l}\frac{\partial p_j^l}{\partial b_j^l} \\ &= \frac{\partial J}{\partial b_j^l}\end{aligned} \tag{2.16}$$

The final backpropagation equation calculates the rate of change of the objective function with respect to an individual weight. This is found through a process similar to that employed to derive Eq. 2.16. To start, the derivative of p_j^l is taken with respect to the individual weight w_{jk}^l. At first this may appear to be the summation of the neuron outputs, $\sum_k o_k^{l-1}$. However, the derivative $\frac{\partial}{\partial w_{jk}^l}$ refers to a specific weight w_{jk}^l, and not the general term for all of the weights in the summation – the use of k under the summation can be misleading.

$$\frac{\partial p_j^l}{\partial w_{jk}^l} = o_k^{l-1} \tag{2.17}$$

The desired equation is found by applying the chain rule and rearranging the scalars on the right hand side.

$$\begin{aligned}\frac{\partial J}{\partial w_{jk}^l} &= \frac{\partial J}{\partial p_{jk}^l}\frac{\partial p_{jk}^l}{\partial w_{jk}^l} \\ &= o_k^{l-1}\delta_j^l\end{aligned} \tag{2.18}$$

In summary, backpropagation is the use of chain rule to calculate the partial derivatives of the objective function with respect to each of the weights and biases of the neural network. The exact steps are explained in Algorithm 2. Next, the parameters of the neural network must be updated using a gradient descent algorithm such as Adam optimizer. This is explained fully in the following section.

Algorithm 2 Backpropagation

procedure BACKPROP($J(x)$, A neural network)
  $\delta^L \leftarrow \nabla_o J \odot \sigma'\left(p^L\right)$ ▷ Calculate the Output Error using Eq. 2.8
  for $l = L-1, L-2, \ldots, 2$ **do** ▷ $l = 1$ is the input vector ξ
    $\delta^l \leftarrow \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(p^l\right)$ ▷ Backpropagate the error using Eq. 2.14
    $\frac{\partial J}{\partial w^l_{jk}} \leftarrow o^{l-1}_k \delta^l_j$
    for all j, k **do**
      $\frac{\partial J}{\partial w^l_{jk}} \leftarrow o^{l-1}_k \delta^l_j$
      $\frac{\partial J}{\partial b^l_j} \leftarrow \delta^l_j$
    end for
  end for
end procedure

2.1.3 Gradient Descent and Adam Optimizer

The process of training a neural network is an optimization problem and can be solved using methods that exist in that field. One of the most common methods to maximize or minimize an objective function is by employing gradient descent. Given an objective function that is differentiable with respect to the variables to be optimized, the gradient is repeatedly calculated and the variables are modified in the opposite direction of, and proportional to, the gradient from the current point. This process ends when the gradient is equal to zero [35]. This will lead to at least a locally optimal solution based on the data used to calculate the gradient, with this data known collectively as a batch. When employing batch gradient descent, the entirety of a data set will usually be used as the batch.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.19}$$

Batch gradient descent can be slow, faces memory issues with large datasets, and is unable to accept new information into the dataset as it becomes available. Within RL, this is an unacceptable limitation. Stochastic gradient descent (SGD) is a variation of this method that is appropriate for neural network optimization. Instead of calculating a single gradient from a large batch of data, it randomly selects a single data point from the entire data set and performs an update. With the gradient

calculations no longer requiring the entire dataset, new data can easily be added as an agent collects it. If an ANN takes data point ξ as an input and is optimized to produce ψ as an output, the SGD update step is as follows.

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; \xi^i; \psi^i\right) \tag{2.20}$$

The use of single data points to calculate gradients makes individual updates noisy. This increases the likelihood that the optimizer will jump out of a sub-optimal basin into another, thus increasing the chance of finding a global optimum. However, this noisiness also increases the difficulty of converging to the global minimum. To mitigate this, batch and stochastic gradient descent are combined to form mini-batch gradient descent. In this method, a small, randomly selected batch of data from the larger, complete dataset is used to calculate each gradient update [54].

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; \xi^{i:i+n}; \psi^{i:i+n}\right) \tag{2.21}$$

SGD optimization is particularly slow in cases where the error basin is long and narrow. Consider a second order optimization problem with an error surface in the shape of an ovular basin: $J(a, b) = a^2 + 5b^2$ as shown in Figure 2-2. By taking a fixed step η in the direction of the gradient, SGD tends to oscillate back in forth along the semimajor axis. Qian [50] demonstrated that by adding a momentum term based upon the gradient from the previous mini-batch, these oscillations could be dampened. For gradient calculation and parameter update u, the update rule with momentum calculates a weighted sum of the previous gradient v_{u-1} with the current gradient $\nabla_\theta J(\theta)$.

$$\begin{aligned} v_u &= \zeta v_{u-1} + \eta \cdot \nabla_\theta J(\theta) \\ \theta &= \theta - v_u \end{aligned} \tag{2.22}$$

One drawback of a momentum based method is that the momentum of the gradient

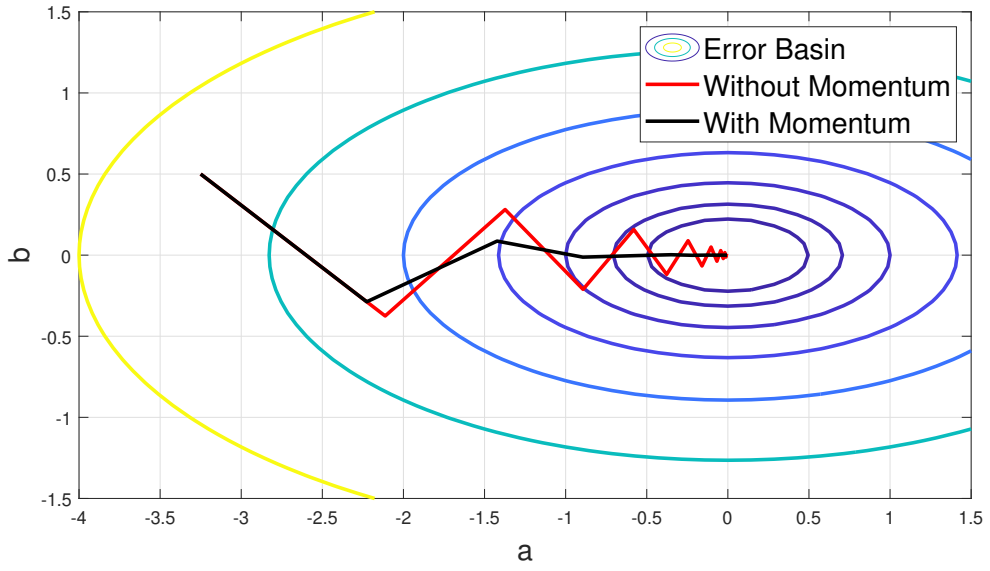

Figure 2-2: Gradient Descent with and without Momentum

builds blindly without regard for the shape of the basin. Additionally, while multiple parameters are being updated, all of the aforementioned methods utilize the same gradient step size. Duchi, Hazan, and Singer developed Adagrad [17] in order to overcome this. In this algorithm, each parameter θ_h in θ receives its own learning rate for each update u, rather than a consistent and uniform value of η. To achieve this, the algorithm defines a diagonal matrix $G_u \in \mathbb{R}^{d \times d}$ for an optimization problem with d parameters. In this matrix, each of the diagonal terms is the sum of the outer product of the gradients from all previous time steps.

$$G_u = \sum_{\tau=1}^{u} \mathbf{g}_\tau \mathbf{g}_\tau{}^T \tag{2.23}$$

With the gradient during update u for each parameter θ_h defined as $g_{u,h}$ in Eq. 2.24, the Adagrad update equation then becomes Eq. 2.25, with the added term κ ensuring that division by zero is impossible.

$$g_{u,h} = \nabla_{\theta_u} J\left(\theta_{u,h}\right) \tag{2.24}$$

$$\theta_{u+1,h} = \theta_{u,h} - \frac{\eta}{\sqrt{G_{u,hh} + \kappa}} \cdot g_{u,h} \tag{2.25}$$

One weakness of Adagrad is that the summation in Eq. 2.23 grows unbounded, eventually causing the the learning rate in Eq: 2.25 to become infinitesimally small [54]. RMSprop [67] is an unpublished neural network optimization algorithm that was developed to solve this issue. It calculates the running average of the gradient, g_u^2, for update u as only a function of the previous update's average and the newly calculated gradient. The weighting coefficients of 0.9 and 0.1 are recommended values.

$$\begin{aligned} \mathbb{E}\left[g^2\right]_u &= 0.9\mathbb{E}\left[g^2\right]_{u-1} + 0.1g_u^2 \\ \theta_{u+1} &= \theta_u - \frac{\eta}{\sqrt{\mathbb{E}\left[g\right]_u + \kappa}} g_u \end{aligned} \tag{2.26}$$

Finally, the optimization algorithm used in this work is reached: Adaptive Moment Estimation (Adam) [34]. A combination of Adagrad and RMSprop, Adam has become one of the most popular optimization methods in RL research. Much as RMSProp and Adagrad used some form of momentum, Adam uses a momentum-like calculation for both the mean m_u and the uncentered variance v_u of the gradient.

$$\begin{aligned} m_u &= \beta_1 \cdot m_{u-1} + (1 - \beta_1) \cdot g_u \\ v_u &= \beta_2 \cdot v_{u-1} + (1 - \beta_2) \cdot g_u^2 \end{aligned} \tag{2.27}$$

The uncentered nature of this estimate is important. These estimates are initialized to zero, and thus are biased towards that value. This is especially relevant early in training, when only a few optimization steps have occurred. To correct for this, bias correction is necessary.

Algorithm 3 *Adam* Optimizer

procedure ADAM(α,β_1,β_2,$F(\theta)$,θ_0)
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
  $u \leftarrow 0$
  while θ_u not converged **do**
    $u \leftarrow u + 1$
    $g_u \leftarrow \nabla_\theta F(\theta_u)$
    $m_u \leftarrow \beta_1 \cdot m_{u-1} + (1 - \beta_1) \cdot g_u$
    $v_u \leftarrow \beta_2 \cdot v_{u-1} + (1 - \beta_2) \cdot g_u^2$
    $\hat{m}_u \leftarrow m_u / (1 - \beta_1^u)$
    $\hat{v}_u \leftarrow v_u / (1 - \beta_2^u)$
    $\theta_u \leftarrow \theta_{u-1} - \alpha \cdot \hat{m}_u / \left(\sqrt{\hat{v}_u} + \epsilon\right)$
  end while
  return θ_u
end procedure

$$\begin{aligned} \hat{m}_u &= \frac{m_u}{1 - \beta_1^u} \\ \hat{v}_u &= \frac{v_u}{1 - \beta_2^u} \end{aligned} \tag{2.28}$$

The parameter update step is then accomplished using a form identical to that of Adagrad and RMSprop.

$$\theta_u = \theta_{u-1} - \alpha \cdot \frac{\hat{m}_u}{\sqrt{\hat{v}_u} + \epsilon} \tag{2.29}$$

This is all summarized in Algorithm 3. The algorithm requires a large number of hyperparameters, which are settings provided by the user that control the behavior of the learning process. For exponential decay rates β_1 and β_2, default values of 0.9 and 0.999 are recommended, respectively. For the term to prevent a divide by zero, ϵ, 10^{-8} is suggested. The step size α is referred to as the learning rate in RL research. The author used values of 0.0001 and 0.0005 for the policy (actor) and critic neural networks, respectively. The terms actor and critic will be defined in Section 2.3.

2.2 Reinforcement Learning Preliminaries

The two fundamental ideas behind Reinforcement Learning is for the computer learner, known as an agent, to learn how to solve a problem by interacting with the world around it, also known as an environment. During these interactions, the agent receives feedback that helps inform it as to whether an action was conducive to solving the problem. By repeatedly attempting to solve the problem and collecting this feedback, the agent eventually learns the solution. This learning process is entirely unsupervised; no training set of correct decisions is provided to the agent and no guidance is given by the researcher other than the feedback signal for actions taken. This provides a number of advantages. If the solution to the problem is unknown, the agent will determine the solution on its own. Since the agent is not being provided with an example of the correct solution, the agent is able to invent novel solutions free from the biases of known solutions generated by traditional methods.

2.2.1 Markov Decision Processes

A more formal description of RL must start with the Markov decision process (MDP), which provides the underlying structure for the learning process [66]. In an MDP, time is subdivided into discrete time steps, $t = 0, 1, 2, \ldots$. The full description of the agent necessary to define its condition within the environment is known as its state, $S_t \in \mathcal{S}$. The action that the agent takes to try and solve the problem at time step t is $A_t \in \mathcal{A}(s)$. Notice how the chosen action is a function of the state. After the agent selects an action, the environment returns a subsequent state S_{t+1} and reward R_t. This state then becomes the current time step as the agent advances into the subsequent time step. This is usually achieved by integrating the dynamics of the agent. This cyclical process is illustrated in Fig. 2-3.

This process continues until some end condition is met. They agent may have succeeded in its task, met a user defined failure condition, or simply reached a pre-set limit on the number of time steps. Each attempt at solving the problem is known in RL as an episode. Many tens or even hundreds of thousands of episodes are

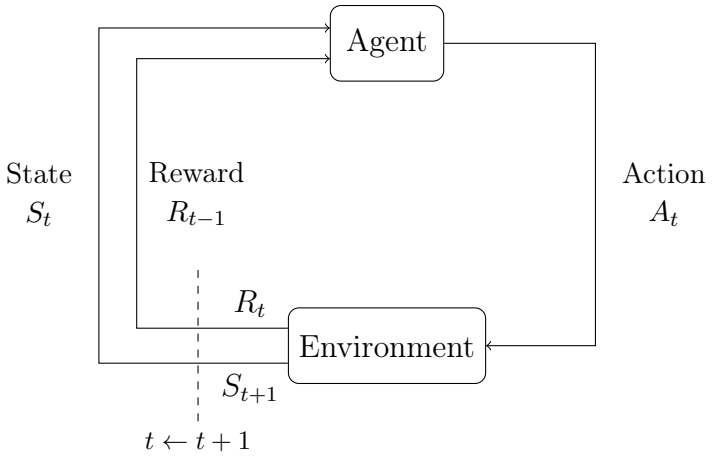

Figure 2-3: Agent - Environment Interaction [66]

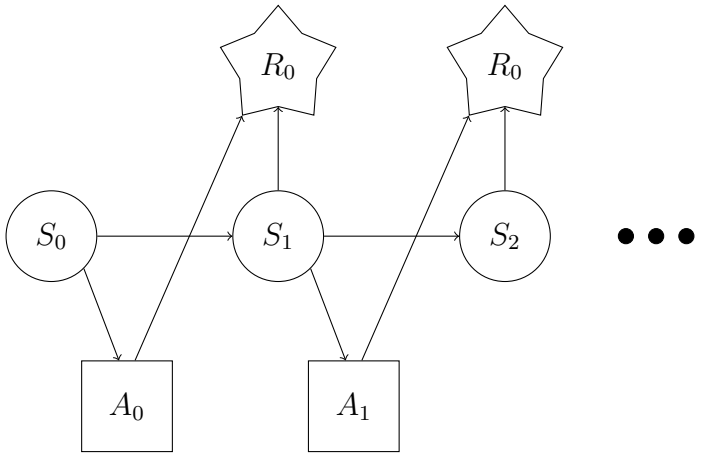

Figure 2-4: State-Action-Reward Trajectory

usually required to train an agent. The sequence of states, actions, and rewards from a completed episode is recorded by the agent and will provide the information necessary to update the agent later on. The trajectory generated during an episode by the cyclical process demonstrated in Figure 2-3 is visualized in a linear manner in Figure 2-4.

Common formulations of RL as MDPs make the assumption that they are finite MDPs. This requires that the sets $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ all have discrete, finite spaces. As a result, the random variables for the state and reward at a time step t, S_t and R_t, have discrete probability distributions that are functions of the state and action at the previous time step [66]. This provides the most important property of an MDP: the

Markov Property. When S_t has the value $\mathbf{s}$ and time t, its future state is unrelated to the past states [75]. In other words, an agent's state at any given time step is only a function of its immediately preceding state. Such a state is referred to as markovian.

Since discrete probability distributions can be defined for S_t and R_t and the state is markovian, it is possible to define the probability of the agent transitioning to a particular state and collecting a given reward solely as a function of the current state and the action associated with it. This function $p\left(\mathbf{s}', r|\mathbf{s}, \mathbf{a}\right)$ is known as the dynamics of the MDP [66] and is a conditional joint probability mass function.

$$p\left(\mathbf{s}', r|\mathbf{s}, \mathbf{a}\right) \doteq Pr\{S_{t+1} = \mathbf{s}', R_{t+1} = r|S_t = \mathbf{s}, A_t = \mathbf{a}\} \tag{2.30}$$

Using Eq. 2.30, other useful values can be calculated. One of the most important is the state-transition probability. This provides the probability of a state at the next time step given a state and action at the current time step. The marginal probability of $S = \mathbf{s}'$ can be found by summing the joint probability of Eq. 2.30 over all possible rewards in the set $\mathcal{R}$, thus producing a conditional probability mass function for S_{t+1}.

$$p\left(\mathbf{s}'|\mathbf{s}, \mathbf{a}\right) \doteq Pr\{S_{t+1} = \mathbf{s}'|S_t = \mathbf{s}, A_t = \mathbf{a}\} = \sum_{r \in \mathcal{R}} p\left(\mathbf{s}', r|\mathbf{s}, \mathbf{a}\right) \tag{2.31}$$

Similarly, the expected reward given a state and action can be found using Eq. 2.30 and the definition of expected value. The expected value of a finite and countable random variable is defined as the sum of all possible values of that variable weighted by their probabilities. For a random variable X with k possible values, $\mathbb{E}[X] = \sum_i^k x_i p_i$.[1] Therefore, the expected value of r is given by

$$r\left(\mathbf{s}, \mathbf{a}\right) \doteq \mathbb{E}\left[R_t|S_t = \mathbf{s}, A_t = \mathbf{a}\right] = \sum_{r \in \mathcal{R}} r \sum_{\mathbf{s}' \in \mathcal{S}} p\left(\mathbf{s}', r|\mathbf{s}, \mathbf{a}\right) \tag{2.32}$$

Notice how the reward is a function of both a state and an action. While an RL researcher will frequently program the reward function within the environment's code as a function of the agent's state, $\mathbf{s}_{t+1}$, the transition to that state is a function of

[1] The use of X, x, i, and k are unrelated to their usage elsewhere in the thesis.

the previous state, $\mathbf{s}_t$, and action, $\mathbf{a}_t$ that brought it there. Therefore the reward is frequently defined as a function of these two variables, as shown here.

As mentioned earlier, this formulation is designed solely for discrete state, action, and reward spaces. This is a critical limitation. In the research presented in Chapters 3 and 4, all of these spaces will be continuous, but bounded. Fortunately, it is not difficult to modify Eqs. 2.31 and 2.32 for continuous random variables. Recall Eq. 2.31, which gave the probability of a specific state $S_{t+1} = s'$. This was a conditional probability mass function. For the continuous case, this is replaced by a conditional probability density function $T(\mathbf{s}, \mathbf{a}, \mathbf{s}')$.

$$p\left(\mathbf{s}'|\mathbf{s}, \mathbf{a}\right) \doteq T\left(\mathbf{s}, \mathbf{a}, \mathbf{s}'\right) \tag{2.33}$$

However, one difference between discrete and continuous random variables is that it is impossible to find the probability of a continuous random variable taking a specific value. As such, Eq. 2.33 is not a one-for-one replacement for Eq. 2.31. Rather, as a probability density function, it can be integrated to calculate the probability that the state will transition to a state within the set $\mathcal{S}'$ [70].

$$\int_{\mathcal{S}'} T\left(\mathbf{s}, \mathbf{a}, \mathbf{s}'\right) d\mathbf{s}' = Pr\left(S_{t+1} \in \mathcal{S}' | S_t = \mathbf{s}, A_t = \mathbf{a}\right) \tag{2.34}$$

For an RL problem with a discrete action space, an action can be selected by simply selecting the value for a_t that has the highest probability from Eq. 2.31. With a continuous action space, an action can be sampled from the probability distribution defined by Eq. 2.33. Regarding the reward function of Eq. 2.32, it would be possible to calculate a continuous version using a probability density function, however this is not necessary in a practical implementation of RL, as shall shown later.

It is also necessary to briefly discuss non-stationary, that is, time-dependent, MDPs. Unlike planning a journey between two cities on a map, it is unfortunately the nature of astrodynamics to cause bodies of interest to move as a function of time, thus complicating any trajectory planning problem. At its most basic, a non-stationary MDP can be defined as a stationary MDP whose state includes time [37]. It has

also been found experimentally that RL algorithms are fairly robust and can handle mildly non-stationary problems where the time-dependent nature of the problem repeats with each episode. For example, Chu, Alfriend, Zhang, and Zhang demonstrated that an RL agent could learn to navigate a field of moving satellites in which the motion of the obstacle spacecraft was consistent in each episode [10]. In the work presented in this thesis, time will be included in the state for any non-stationary problem.

2.2.2 The Policy, Value, and Advantage Functions

With the discussion of the underlying MDP theory completed, more practical details for the implementation of RL can be delved into. First, a note regarding notation. Thus far, an agent has been said to take an action $A_t = \mathbf{a}_t$ from the set $A_t \in \mathcal{A}$ while at the state $S_t = \mathbf{s}_t$ from the set $S_t \in \mathcal{S}$. This shall now be streamlined to simply say that the agent takes the action $\mathbf{a}_t$ while at state $\mathbf{s}_t$. Note the use of bold letters to mark these as vectors.

An agent is not a monolothic system, but rather contains subcomponents that control its various functions. When an agent chooses an action, it does so according to its policy π. This policy maps an observation of the agents environment $\mathbf{o}_t$ to an action $\mathbf{a}_t$ and transforms the agent into a closed-loop controller. For continuous problems such as those presented here, this policy is a neural network that takes the observation input signal $\mathbf{o}_t$ and defines a median and standard deviation as an output in order to define a probability distribution. This is written as $\pi_\theta\left(\mathbf{a}_t|\mathbf{o}_t\right)$, where θ are the parameters of the neural network. The distinction between observation and state is a subtle one. While the input signal to the policy should satisfy the Markov property, in many practical applications it does not. In some scenarios, a researcher may determine that the observation should contain more information than is included in the state alone. Information derived from the base state may be provided to the agent. In this investigation, these complications shall be removed by making the assumption that the MDP is fully observable and that the state and observation are one and the same. The policy becomes $\pi_\theta\left(\mathbf{a}_t|\mathbf{s}_t\right)$.

In the previous section, the cyclical nature of the MDP formulation was discussed. Each episode starts from an initial state provided by the environment, $\mathbf{s}_0$. The agent selects an action, $\mathbf{a}_0$ according to its policy $\pi_\theta\left(\mathbf{a}_0|\mathbf{s}_0\right)$ and communicates this action to the environment. The agent receives, in response, the subsequent state s_1 and a reward r_0 based upon that state. The agent then records these selections as a tuple $(\mathbf{s}_0, \mathbf{a}_0, r_0)$ and repeats the cycle until reaching an end condition or a pre-determined final time step. In doing so, a full trajectory of these tuples is recorded. After some user-selected number of episodes, the policy is updated based on the information contained with these tuples. After many updates, the agent eventually converges to an optimal policy, π^*. Learning this optimal policy, however, will require more than just the reward associated with each state-action pair. A new concept must now be introduced: the value of a state, $V(\mathbf{s}_t)$.

In order to make intelligent decisions, an agent must consider not only the immediate consequences of its actions but also their long term ramifications. With reward being a function of only a single state transition from $\mathbf{s}_t$ to $\mathbf{s}_{t+1}$ via $\mathbf{a}_t$, determining decisions in a greedy manner by selecting the action with the highest reward could result in a poor policy. To correct this deficiency, the value of a state is defined as the expected value of the sum of discounted future rewards. Assuming a final time step T and a discounting factor $\gamma < 1$,

$$V(\mathbf{s}_t) = \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T\right] = \mathbb{E}\left[\sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau\right] \tag{2.35}$$

The value can be thought of as being a measure of the quality of a trajectory, with the horizon of this estimate controlled by the discounting factor. The discounting factor also controls the greediness of the agent, or the extent to which it prioritizes the immediate reward over the value of the complete trajectory.

$$V(\mathbf{s}_t) = \mathbb{E}\left[r_t\right] + \gamma V(\mathbf{s}_{t+1}) \tag{2.36}$$

With $\gamma = 0$, the agent would gauge the quality of a state based purely on the immediate reward it can expect to immediately receive according to its policy. After an episode is completed, the values of each state in that exact trajectory can be calculated recursively using Algorithm 4.

There remains one more function to define. During training, the agent is exploring its environment and trying many actions. In order to improve its policy, it must determine whether a given action leads to a better or worse outcome than it anticipated, with regard to solving the provided problem. Assume that the agent has a means of estimating the value function. This represents the agent's best guess as to the value of each state according to its current policy. As such, this will now be noted as $V^{\pi}(\mathbf{s}_t)$. If the agent's policy is the optimal policy π^*, then there should be no trajectory starting from state $\mathbf{s}_t$ that is greater than $V^{\pi^*}(\mathbf{s}_t)$.

Upon completing an episode, the agent calculates the true value of each state that it encountered using the collected rewards against its current value function. This is known as the advantage function. As detailed by Shulman et al. [57], it is defined as

$$A^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) = Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) - V^{\pi}\left(\mathbf{s}_t\right) \tag{2.37}$$

The first term, $Q\left(\mathbf{s}_t, \mathbf{a}_t\right)$ is the state-action value function. It is an estimate of the value of the trajectory starting from state $\mathbf{s}_t$ following the selection of action $\mathbf{a}_t$. Since the agent is exploring the environment, this will likely not be the best action according to its policy, but rather one that it is trying. The second term, is, of course, the value function. Should the advantage be positive, then the selection of action $\mathbf{a}_t$ resulted in a trajectory that was superior to the agent's current policy. As such, it should be changed accordingly.

In practice, the advantage is computed by comparing the true values of each state, $\mathfrak{V}$, as calculated by Algorithm 4, against the agent's current value function. This is outlined in Algorithm 5.

$$A^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) = \mathfrak{V}_t^{\pi} - V^{\pi}\left(\mathbf{s}_t\right) \tag{2.38}$$

Algorithm 4 Value Calculation for a Completed Episode

procedure VALUE CALCULATION(rewards, γ)
  Note: rewards is the list $[r_0, r_1, \ldots, r_T]$
  $T \leftarrow \text{length(rewards)} - 1$ ▷ Grab the index of the final state
  $\mathfrak{V}_T \leftarrow r_T$ ▷ r_T may be zero if $\mathbf{s}_{T+1}$ cannot exist
  for $t = T-1, ..., 0$ **do**
    $\mathfrak{V}_t \leftarrow r_t + \gamma \mathfrak{V}_{t+1}$ ▷ Based on Eq. 2.36
  end for
  return $\mathfrak{V}$ ▷ Returns a list of values for each state as visited in an episode
end procedure

Algorithm 5 Advantages Calculation for a Completed Episode

procedure ADVANTAGE CALCULATION(states, $\mathfrak{V}$, V)
  Note: states is the list $[\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_T]$
  for $t = 0, \ldots, T$ **do**
    $A^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \mathfrak{V}_t - V(\mathbf{s}_t)$
  end for
  return A^{π} ▷ return an list of advantages for for each state in this episode
end procedure

2.3 The Actor-Critic Framework

Within RL, algorithms are frequently broadly classified as either value optimization or policy optimization methods. In the former, the agent attempts to learn the optimal value function and to derive a policy from this knowledge. This is often accomplished by iterating on the Bellman equation [73] which states that the optimal value function is that which corresponds to a policy that returns the maximum lifetime reward.

$$V^*(\mathbf{s}_t) = \max_{\mathbf{a}_t \in \mathcal{A}} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^*(T(\mathbf{s}_t, \mathbf{a}_t)) \right] \tag{2.39}$$

With sufficient time, this could be found by repeatedly visiting every possible state $\mathbf{s}$ using every possible transition to that state T until the value function converges. After completing this process of iteration, the optimal policy can be trivially found using

$$\pi^*(\mathbf{s}_t) = \arg\max_{\mathbf{a}_t \in \mathcal{A}} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^*(T(\mathbf{s}_t, \mathbf{a}_t)) \right] \tag{2.40}$$

Value optimization algorithms are sometimes referred to as critic-only methods, for they are based purely on judging the value of each state. Unfortunately, they are computationally expensive due to the requirement of visiting every possible state. This is especially true if the action space is likewise continuous. As a result, the action space is usually discretized [29].

Policy optimization methods, also known as actor-only methods, take the opposite approach in that it seeks to directly optimize the policy and then back the optimal value function out of it. It does so through an iterative process. Starting from an initial policy, a corresponding value function is computed, which in turn is used to produce an improved policy function. This process repeats until convergence. Unlike value optimization, this does not require visiting every state and the action space can easily be kept continuous. However, most policy optimization methods belong to the subcategory of algorithms known as policy gradient methods which suffer from slow learning due to high variances in the estimate of gradients [29].

Proximal Policy Optimization, while belonging to the policy gradient family, uses a hybrid framework known as an actor-critic method. Rather than only learn the policy or value function and then back out the other, both are optimized simultaneously. Unlike critic-only methods, the policy's action space can remain continuous. The issue with high variance gradients in actor-only methods is reduced by having the critic provide lower-variance estimates of the value of each state. While not perfect, the actor-critic framework have good convergence properties and are a popular structure for designing RL algorithms [29].

2.3.1 A Note on the Practical Implementation of Actor-Critic Methods

To implement an actor-critic RL algorithm, two neural networks are required: an actor and a critic. The actor ANN produces the policy, $\pi(\mathbf{a}_t|\mathbf{s}_t)$ while the critic calculates the agent's best estimate of the value function, $V(\mathbf{s}_t)$. Upon completing some number of episodes – the rewards collected from many episodes may be batched together –

the two networks must be updated. The loss function for the policy depends on the algorithm, but it will generally contain the advantage function, Eq. 2.38. If the advantage is found to be zero, then the agent has converged onto what it believes to be the optimal policy. In order to update the critic, the loss function is the difference between the true values calculated using Algorithm 4 and the critic's assessment, $V(\mathbf{s}_t)$. That is, it is trying to optimize the critic in order to minimize Eq. 2.38.

2.4 Policy Gradient Methods

Policy gradient methods are a class of RL algorithms that optimize the parameters that compose a policy ANN using a gradient descent method. Consider an agent that determines its actions by sampling from a policy defined by a normal distribution, $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$. Through the use of gradient descent, the policy will be optimized in order to maximize the expected sum of discounted rewards, $\sum_{t=0}^{T} \gamma r_t$. The gradient is thus defined as

$$g \doteq \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{T} \gamma r_t\right] \tag{2.41}$$

According to Schulman et a. [57, 58], there are many similar expressions for the gradient, but for the purposes of this explanation, it shall be defined as

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta\left(\mathbf{a}_t|\mathbf{s}_t\right) \hat{A}_t\right] \tag{2.42}$$

where π_θ is a stochastic policy and $\hat{A}_t$ is an estimate of the advantage function. With regards to the earlier discussion of gradients in Section 2.1, this refers to the gradient for the entire neural network during a given update, and not that of any single layer, neuron, or parameter.

Recalling equations such as Eq. 2.19 and 2.20 from Section 2.1.3, the gradient should be the derivative of the objective function. Therefore, the objective function corresponding to Eq. 2.42 is

$$J^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta \left(\mathbf{a}_t | \mathbf{s}_t \right) \hat{A}_t \right] \tag{2.43}$$

Unfortunately, repeatedly performing optimization on J^{PG} using the same state-action-reward trajectory has been shown to lead to destructively large policy updates [58]. To address this, Schulman et al. developed Trust Region Policy Optimization (TRPO) as an algorithm with guaranteed monotonically improvement [56]. This is achieved by limiting the change in the policy to small updates. It is often likened to climbing a mountain along a narrow ridge. Rather than take a large step along the direction of the gradient and risk stepping off the ridge, it is far safer to take many smaller steps.

In TRPO, episodes are completed according to the policy $\pi_{\theta_{\text{old}}}$. To optimize the policy, the collected state-action-reward data is used to optimize a second, identical policy ANN, π_θ. In the spirit of the aforementioned mountain climbing example, a limit is placed on the difference between the newly optimized policy and the original policy under which the episodes were completed. This is accomplished using the KL divergence, which measures the difference between probability distributions. The optimization problem is defined as

$$\begin{aligned} \underset{\theta}{\text{mazimize}} \quad & \hat{\mathbb{E}}_t \left[R_t(\theta) \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}} \left(\cdot | \mathbf{s}_t \right), \pi_\theta \left(\cdot | \mathbf{s}_t \right) \right] \right] \leq \zeta \end{aligned} \tag{2.44}$$

using the policy ratio R_t of

$$R_t = \frac{\pi_\theta \left(\mathbf{a}_t | \mathbf{s}_t \right)}{\pi_{\theta_{\text{old}}} \left(\mathbf{a}_t | \mathbf{s}_t \right)} \tag{2.45}$$

In TRPO, this ratio is the probability of an action occurring under the new policy, π_θ, divided by the probability as it occurred under the previous policy, $\pi_{\theta_{\text{old}}}$. The algorithm attempts to maximize $R_t(\theta)\hat{A}_t$, or the product of the ratio for a given action $\mathbf{a}_t$ and the advantage calculated for that action. When an action leads to a

beneficial outcome, the policy is optimized to make that outcome more likely, thus making $R_t > 1$. Such an action would also have a positive advantage, making $R_t(\theta)\hat{A}_t$ positive as well. If the action had a detrimental result, the action should be made less likely: $R_t < 1$. Since the advantage function for an undesirable action is negative, the maximization in Eq. 2.44 will produce the least negative $R_t(\theta)\hat{A}_t$ possible. In doing so, it will making the probability of the action occurring under the new policy, π_θ as small as possible, subject to the constraint on the KL divergence.

2.5 Proximal Policy Optimization

While TRPO is a powerful algorithm, it is also complex, among other limitations. Proximal Policy Optimization (PPO) was developed by Schulman et al. as a simpler algorithm while maintaining the same high-level of performance [58]. In PPO, the ratio-advantage product is carried over from TRPO. However, the KL divergence constraint is removed and integrated into the objective function itself.

$$J^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(R_t(\theta)\hat{A}_t, \text{clip}\left(R_t(\theta), 1-\epsilon, 1+\epsilon\right)\right)\hat{A}_t\right] \tag{2.46}$$

Consider an action that resulted in a better than expected outcome. In such a case, the optimization seeks to make this outcome more likely, with $R_t(\theta) > 1$. To avoid any overly changes to the policy, the change in probability is limited to a value ϵ. For example, if $\epsilon = 0.2$, $R_t(\theta)$ may not exceed 1.2 and the action may not become more than 20% more likely. The limit applies in the opposite direction for undesirable actions; the likelihood may not be reduced below 80% of its previous value.

While the clipped version of PPO is the most common implementation, other versions exist. One such method brings back the KL divergence. In this version, the objective function to be maximized contains a second, KL divergence term.

$$J^{KLPEN}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta\left(\mathbf{a}_t|\mathbf{s}_t\right)}{\pi_{\theta_{\text{old}}}\left(\mathbf{a}_t|\mathbf{s}_t\right)}\hat{A}_t - \beta\text{KL}\left[\pi_{\theta_{\text{old}}}\left(\cdot|\mathbf{s}_t\right), \pi_\theta\left(\cdot|\mathbf{s}_t\right)\right]\right] \tag{2.47}$$

Rather than control the change in policy with a clipping factor, the optimizer aims to maintain a KL divergence d of d_{targ} by actively modifying the coefficient β. Should the change in policy be too great and $d > d_{targ}$, β is increased in order to encourage smaller changes. The opposite holds true as well, with β decreasing should $d_{targ} > d$. The work presented in the following chapters utilized this final version of PPO, along with minor alterations based on Patrick Coady's open source work.[2]

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

Chapter 3

Earth-Mars Mass-Optimal Transfer

In this chapter, an agent will be trained to complete a low-thrust, mass-optimal transfer between the Earth and Mars. By analyzing the results, the optimality and accuracy of PPO is compared to a traditional solver will be determined. Two cases will be presented. In the first, fixed-time steps of equal length will be used with the control law (action) held constant for the length of each time step. The second case will allow the agent to control the length of each time step and thus determine when to change its control policy. These results were initially presented at the 2019 AAS Astrodynamics Specialist Conference [41].

3.1 Problem Formulation

The desired transfer is a fixed-length, 302 day mission composed of 22 time steps, each of which has its own state. Between states, an action composed of a thrust direction and magnitude is held constant and the dynamics are integrated forward to determine the subsequent state. Initial conditions are provided by first solving the problem using EMTG's high-fidelity Parallel Shooting with Finite Burn (PSFB) mode, which is capable of also determining the first post-launch state. This solution is considered to be the baseline to compare against. The initial state vector can be found in Table B.1. Mars ephemeris data was provided by SpiceyPy [3], a Python wrapper for the SPICE toolkit [2, 1] provided by NASA's Navigation and Ancillary

Information Facility (NAIF). The start time for the mission is 2,454,396.30662312 JD (Julian Date).

In Case 1, a fixed integration time between states is used that separates each time step by an equal amount of time. This is how RL control problems are normally defined. The variable integration length used in Case 2 was to provide the agent with an additional degree of freedom that would let it behave more like an optimal control optimizer, which will often use unequal spacing in its solution's mesh. If successful, the agent will place more time steps in portions of the trajectory solution that require finer control.

3.1.1 Dynamics

The dynamics were modelled as a two-body problem in the sun-centered inertial frame. Control was applied as a continuous and constant thrust magnitude and direction along each integration arc. State integration was accomplished using the *odeint* function contained within the SciPy library [71]. In order to provide the neural network with similarly sized inputs across all variables, the dynamics were nondimensionalized using the values in Table 3.1. The characteristic distance is defined as $L_{sc} = 1\,\text{AU}$. Using this distance and the two-body gravitational constant of μ_{sun}, the velocity and time characteristic values were calculated. Necessary spacecraft specifications are defined in Table 3.2. The mass variable m is non-dimensionalized based on the spacecraft's initial mass m_0 and bounded from 0 to 1. The variable r is the distance between the central body – the Sun – and the spacecraft. The acceleration from the spacecraft's propulsion system in a given direction (x, y, z) is given by multiplying the thrust f with the appropriate unit vector component u_x, u_y, and u_z, respectively.

μ_{sun}	L_{sc}	V_{sc}	T_{sc}	g_0
$1.3271244018 \times 10^{20}\,\text{m}^3\,\text{s}^{-2}$	149,597,870,691 m	$\sqrt{\frac{\mu}{L_{sc}}}$	$\frac{L_{sc}}{V_{sc}}$	$9.80665\,\text{m}\,\text{s}^{-2}$

Table 3.1: Dynamics Constants and Scaling Factors

m_0	I_{sp}	$\max f$
525.2 kg	3,000 s	0.1 N

Table 3.2: Spacecraft Specifications

$$\begin{aligned}
\ddot{x} &= -\frac{x}{r^3} + \frac{f \cdot u_x \cdot T_{sc}^2}{m_0 \cdot m \cdot L_{sc}} \\
\ddot{y} &= -\frac{y}{r^3} + \frac{f \cdot u_y \cdot T_{sc}^2}{m_0 \cdot m \cdot L_{sc}} \\
\ddot{z} &= -\frac{z}{r^3} + \frac{f \cdot u_z \cdot T_{sc}^2}{m_0 \cdot m \cdot L_{sc}}
\end{aligned} \tag{3.1}$$

The rate of change in spacecraft mass was calculated based on the magnitude of the commanded thrust, the spacecraft's current mass, and the I_{sp} of the thruster. The characteristic values for length and time, as well as the initial spacecraft mass, were used to produce a non-dimensionalized rate.

$$\dot{m} = -\frac{\sqrt{u_x^2 + u_y^2 + u_z^2}}{m_0 \cdot m} \cdot \frac{g_0 \cdot T_{sc}^2}{L_{sc}} \cdot \frac{I_{sp}}{T_{sc}} \tag{3.2}$$

3.1.2 RL Implementation

Each episode was composed of 22 time steps and began with the initial conditions calculated by EMTG. If successful, the final time step would place the spacecraft at Mars's position 302 days later. 21 state transitions therefore occurred per episode, each with a corresponding action and reward. At the final state, no future action was calculated and the value was set to zero, since no future states would exist after arrival.

To satisfy the requirements of the Markov property, the agent's state (observation) must provide sufficient information such that the transition to the following state is only a function of the present state. The spacecraft dynamics can be fully defined in Eqs. 3.1 and 3.2 by its position, velocity, and mass: $x, y, z, \dot{x}, \dot{y}, \dot{z}, m$. These accordingly make up the first 7 variables in the state. In order to rendezvous with Mars, the planet's position must also be known at that moment. Rather than give the

agent Mars's actual position and velocity, the error in position and velocity between the spacecraft and Mars was used: $x_{\text{err}}, y_{\text{err}}, z_{\text{err}}, \dot{x}_{\text{err}}, \dot{y}_{\text{err}}, \dot{z}_{\text{err}}$. These were chosen because they appear in the reward function and are based on previous success with a similar orbit transfer problem in [42]. As a fixed time orbital transfer, the agent was expected to arrive at Mars by the 22$^{\text{nd}}$ and final time step. Therefore, the current time step was also provided. Similar to the scaling of the dynamics, t_{scaled} was defined as $\frac{t}{21}$ and thus bounded from 0 to 1. The vector composed of these variables defines the state at any given time step.

$$\mathbf{s}_t = [x, y, z, \dot{x}, \dot{y}, \dot{z}, m, x_{\text{err}}, y_{\text{err}}, z_{\text{err}}, \dot{x}_{\text{err}}, \dot{y}_{\text{err}}, \dot{z}_{\text{err}}, t_{\text{scaled}}]^T \tag{3.3}$$

In Case 1, the agent determined the magnitude f and unit vector components of the thrust vector $\left[u_x, u_y, u_z\right]^T$. Recall that actions were sampled from a multivariate normal distribution defined by the output of the actor NN. The final layer of the actor ANN used a tanh activation function so that the mean of each dimension of the distribution would be between -1 and 1. However the standard deviation of the distribution allows for values outside of this range, so the outputs were clipped element-wise back within this range if necessary. Let a tilde denote the initial ANN output versions of the controls such that $\tilde{f}, \tilde{u}_x, \tilde{u}_y, \tilde{u}_z \in [-1, 1]$.

$$\mathbf{a}_t = \begin{bmatrix} \tilde{f} & \tilde{u}_x & \tilde{u}_y & \tilde{u}_z \end{bmatrix} \tag{3.4}$$

To be used in the equations of motion, the requested thrust $\tilde{f}$ was mapped to the nondimensionalized equivalent of the allowable range of 0 to 0.1 N to produce the thrust equivalent f used in the Eq. 3.1. The direction of the vector was also normalized to produce a unit vector with components u_x, u_y, u_z.

$$\hat{\mathbf{u}} = \left[u_x, u_y, u_z\right] = \frac{\left[\tilde{u}_x, \tilde{u}_y, \tilde{u}_z\right]}{\|\tilde{u}_x, \tilde{u}_y, \tilde{u}_z\|} \tag{3.5}$$

In Case 2, control over integration time was added to the action $\mathbf{a}_t$ with the addition of a new variable $\tilde{\mathfrak{T}}$ appended to the control vector. As with the other control outputs, $\tilde{\mathfrak{T}} \in [-1, 1]$. This was then mapped to a variable $\mathfrak{T}$ between 5% and

Case	c_1	c_2	c_3	c_4	c_5	c_6
Case 1	0.5	-500	-0.05	-250	0.25	0
Case 2	0.5	-250	-0.05	-250	0.025	0.25

Table 3.3: Reward Function Coefficient

100% of the maximum allowable integration time of of approximately 30 d that would be used to advance to the next state. The exact maximum value is provided in Table B.2 within the Appendix.

$$\mathbf{a}_t = \left[\tilde{f}, \tilde{u}_x, \tilde{u}_y, \tilde{u}_z, \tilde{\mathfrak{T}}\right] \tag{3.6}$$

With the integration time no longer constant, the portion of the 302 d mission that has elapsed can no longer be measured by the nondimensionalized time step counter t_{scaled} found in Eq. 3.3. To account for the mismatch between mission time and time steps, an additional variable $t_{\text{elapsed}} \in [0, 1]$ for measuring the the elapsed time was added to the state during Case 2.

$$\mathbf{s}_t = \left[x, y, z, \dot{x}, \dot{y}, \dot{z}, m, x_{\text{err}}, y_{\text{err}}, z_{\text{err}}, \dot{x}_{\text{err}}, \dot{y}_{\text{err}}, \dot{z}_{\text{err}}, t_{\text{scaled}}, t_{\text{elapsed}}\right]^T \tag{3.7}$$

The reward function for both cases was a combination of linear and exponential error terms with weighting controlled by the coefficients found in Table 3.3. Mass optimality and constraints were achieved using a penalty method within the reward function. This function is

$$r_t = -\rho_{t_{\text{pos.}}} - \rho_{t_{\text{vel.}}} + c_1 e^{c_3} + c_2 e^{c_4 \rho_{t_{\text{vel.}}}} - c_5 \left(1 - m\right) - c_6 \mathfrak{p}_{\text{time}} + \mathfrak{p}_{\text{spatial}} \tag{3.8}$$

were the position and velocity error magnitudes are represented as $\rho_{t_{\text{pos.}}}$ and $\rho_{t_{\text{vel.}}}$, respectively.

$$\begin{aligned} \rho_{t_{\text{pos.}}} &= \|x_{\text{err}} \quad y_{\text{err}}\| \\ \rho_{t_{\text{vel.}}} &= \|\dot{x}_{\text{err}} \quad \dot{y}_{\text{err}}\| \end{aligned} \tag{3.9}$$

The penalty term $\mathfrak{p}_{\text{time}}$ is used to enforce the required 302 day transfer in Case 2. It is a linear penalty placed on the final state based on the deviation from the desired mission elapsed time. In Case 1, the fixed-length time step ensures that this condition is met, making the term unnecessary.

$$\mathfrak{p}_{\text{time}} = \begin{cases} \|302 - 302 \cdot t_{\text{elapsed}}\| & \text{if } t_{\text{scaled}} = \frac{20}{21} \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

The spatial penalty term $\mathfrak{p}_{\text{spatial}}$ was included based on experience in previous work [42]. It ends the episode prematurely and applies a large penalty if the spacecraft leaves the region of space in which it should be travelling. These bounds were placed close to the Sun, past the orbit of Mars, and outside of the plane of the orbits of the Earth and Mars. Due to the limited thrust available to the spacecraft, violating these bounds was difficult and this penalty was rarely, if ever, activated during training.

$$\mathfrak{p}_{\text{spatial}} = \begin{cases} -20 \text{ \& done}, & \text{if } \|x, y, z\| < 0.25 \\ -20 \text{ \& done}, & \text{if } \|x, y, z\| > 1.75 \\ -20 \text{ \& done}, & \text{if } \|x, y, z\| > 0.75 \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

Implementing constraints within the reinforcement learning framework is an ongoing challenge, as is designing reward functions more broadly. All of the information that the agent learns from is contained within a single scalar value. If any term in the reward for one objective is too large, it may dominate and the agent will neglect the other goals and constraints. For example, if the mass penalty term is too small, the

agent may apply thrust unnecessarily. If it is too large, the training may converge to a policy that fails to apply sufficient thrust to meet the primary objective of arriving at Mars. Balancing the magnitude of each term in the reward function can only be achieved by tuning the various weighting coefficients through a process of trial and error.

The ANNs for the actor and the critic were each composed of 3 hidden (internal) layers of 32 neurons each. These hidden layers all used ReLU activation functions, while the output layer activation functions depended on the desired product of the network. As has been discussed extensively, the actor network used tanh in the output layer to bound each element of $\mathbf{a}_t$ between -1 and 1. The critic needed to produce a scalar value that was unbounded and could be positive or negative. It made use of a linear function, which is identical to simply not having an activation function at all. Learning rates were reduced over the course of the episode according to the schedules provided in Tables B.3 and B.4. When updating the ANNs, the appropriate learning rate is used as the gradient step size α in Eq. 2.29.

3.2 Results

The following results were produced using Python and TensorFlow on a Lenovo ThinkPad P51 with an Intel Xeon E3-1505M v6 CPU, an NVIDIA Quadro M2200 GPU, and 16 GB of RAM. Training times were on the order of hours.

3.2.1 Case 1: Fixed Time Step

The trajectory produced for the Case 1 scenario is shown in Figure 3-1, with the agent's trajectory in blue, the EMTG solution in yellow, Mars's path in red, and the thrust vectors of the agent displayed as black arrows. The agent coasted for the first half of the transfer and maximum allowable thrust in the second half. Such a bang-bang control law matches that of the EMTG solution, as shown by the nearly identical mass profile found in Figure 3-2.

The position and velocity error with respect to Mars can be seen to decrease over

time in Fig. 3-3. The final position and velocity error for Case 1 can be found in Table 3.4. The overall position and velocity error with respect to Mars are both on the order of 10^{-3} nd (nondimensional units). The position error of 1.8159×10^5 km is just within Mars's 5.78×10^5 km sphere of influence [60], or the distance at which Mars's gravity begins to dominate the dynamics of the spacecraft. 479.4646 kg of mass was delivered, compared to the optimal EMTG value of 481.0995 kg, for an error of 1.6379 kg or 3.1186×10^{-3} nd. For comparison, the EMTG solution had final position and velocity errors on the order of 10^{-9} nd.

It is interesting to note that while the accuracy of the position, velocity, and mass are all similar without units, once dimensionalized, the error varies by a wide range of orders of magnitude. This is logical based on the reward function structure and the use of similarly scaled nondimensional units. The linear position and velocity terms, for example, would be of the same order of magnitude in such a scenario. This suggests that any difference in accuracy achieved by the agent in position and velocity is primarily due to the reward function, as opposed to either error being more difficult than the other to achieve.

		Position Error		
		x	y	z
Components	[nd]	5.7019×10^{-4}	-1.0363×10^{-3}	-2.7277×10^{-4}
	[km]	8.5299×10^{4}	-1.5503×10^{5}	-4.0806×10^{4}
Total [l^2-norm]	[nd]	1.2183×10^{-3}		
	[km]	1.8159×10^{5}		

		Velocity Error		
		$\dot{x}$	$\dot{y}$	$\dot{z}$
Components	[nd]	5.3032×10^{-4}	-7.6738×10^{-4}	4.9067×10^{-4}
	[km s^{-1}]	1.5795×10^{-2}	-2.2856×10^{-2}	1.4614×10^{-2}
Total [l^2-norm]	[nd]	1.0540×10^{-3}		
	[km s^{-1}]	3.1392×10^{-2}		

Table 3.4: Case 1 Final State Error

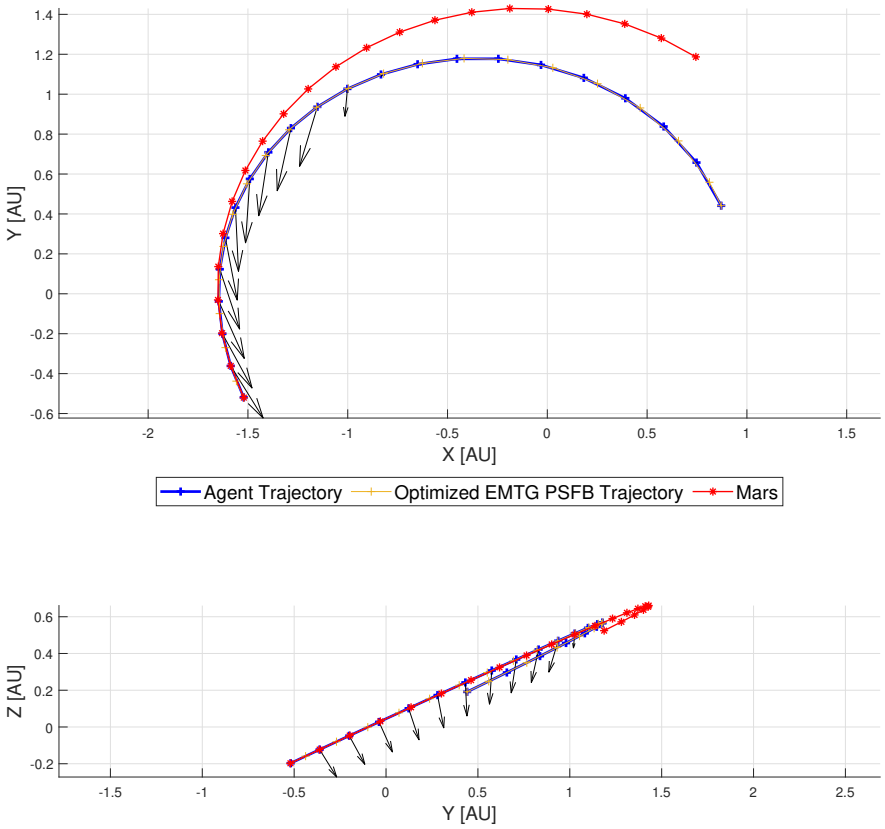

Figure 3-1: Case 1 Trajectory

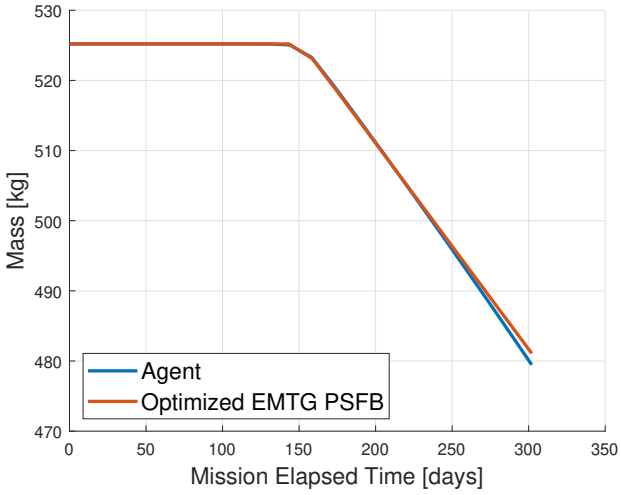

Figure 3-2: Case 1 Mass History

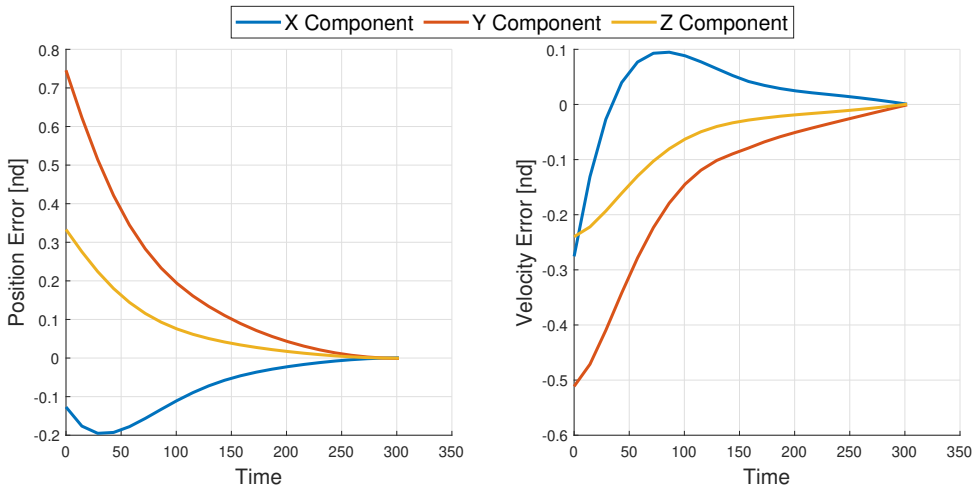

Figure 3-3: Case 1 Error History

3.2.2 Case 2: Variables Time Step

In Case 2, the agent's use of variable length time steps is readily apparent by comparing the spacing of the points in the trajectory shown in Figure 3-4 with that in Figure 3-1 from Case 1. The agent placed more states, and therefore updates to the control law, closer to Mars, rather than during the initial coast phase. This was the expected behavior, since it allows for finer improvements and control updates are unnecessary during coast. However, there is another reason that this behavior may be observed. Since regions closer to Mars provide higher rewards, it is possible that the agent was simply better able to maximize the sum of future rewards (i.e. value) by placing many states at the end of the transfer.

The final position and velocity errors in Table 3.5 were of the same nondimensionalized order of magnitude as in Case 1, with slightly improved position error and marginally worse velocity error. Compared to EMTG, the accuracy is therefore still poor. The decrease in the individual position and velocity error components can be found in Figure 3-5. The time constraint used to enforce the 302 d transfer requirement was effective and provided a final error of approximately 1 hr and 21 min. At an accuracy of 10^{-4}, this is a similar level of performance as the position and velocity. An additional mass of 1.2386 kg was delivered compared to the EMTG solution, sug-

gesting that the agent found a way to save a small quantity of fuel while achieving a similar spatial performance to Case 1. The mass history of the transfer is displayed in Figure 3-6. The burn can be seen to start earlier in the trajectory, but the slope is slightly flatter, indicating that a longer burn of reduced thrust was used.

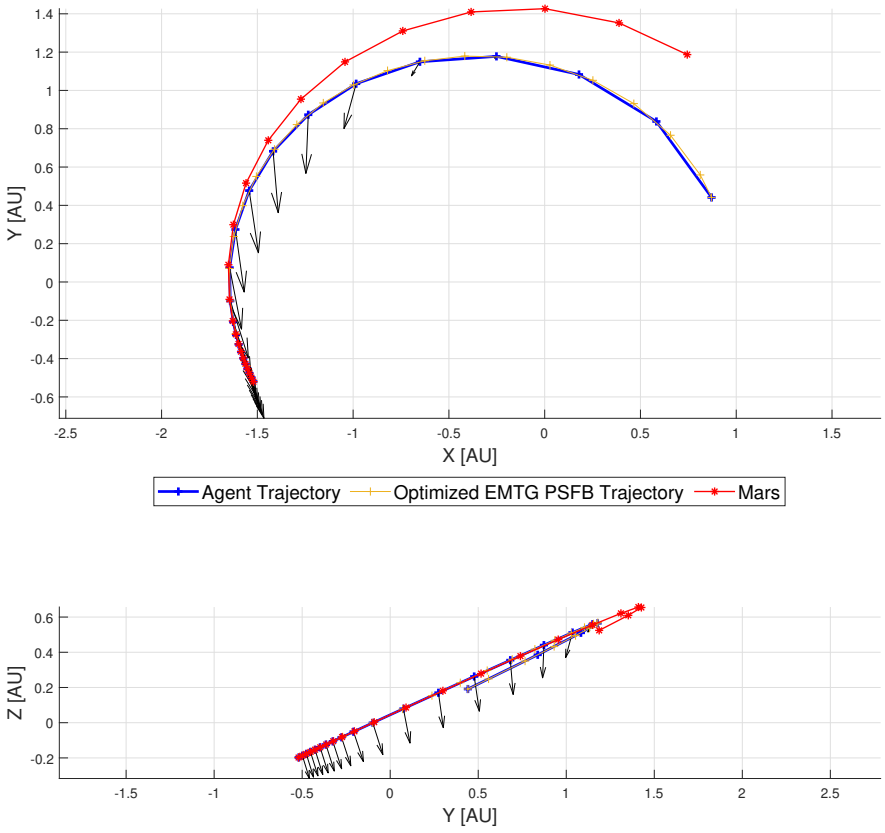

Figure 3-4: Case 2 Trajectory

		Position Error		
		x	y	z
Components	[nd]	6.0574×10^{-4}	-9.4371×10^{-4}	-2.3980×10^{-4}
	[km]	9.0617×10^{4}	-1.4118×10^{5}	3.5874×10^{4}
Total [l^2-norm]	[nd]	1.1467×10^{-3}		
	[km]	1.7155×10^{5}		

		Velocity Error		
		$\dot{x}$	$\dot{y}$	$\dot{z}$
Components	[nd]	4.3802×10^{-3}	-6.9200×10^{-3}	-7.7607×10^{-4}
	[km s^{-1}]	1.3046×10^{-1}	-2.0610×10^{-1}	-2.3115×10^{-2}
Total [l^2-norm]	[nd]	8.2263×10^{-3}		
	[km s^{-1}]	2.4502×10^{-1}		

Table 3.5: Case 2 Final State Error

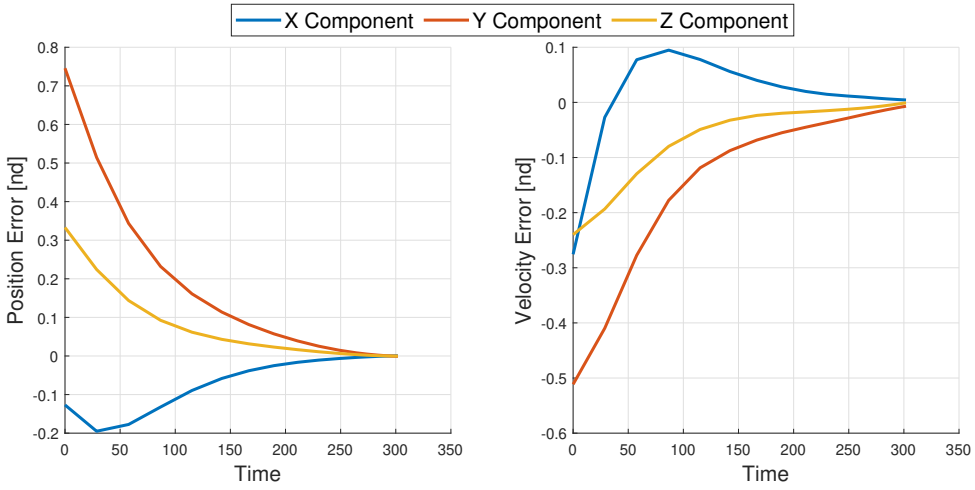

Figure 3-5: Case 2 Error History

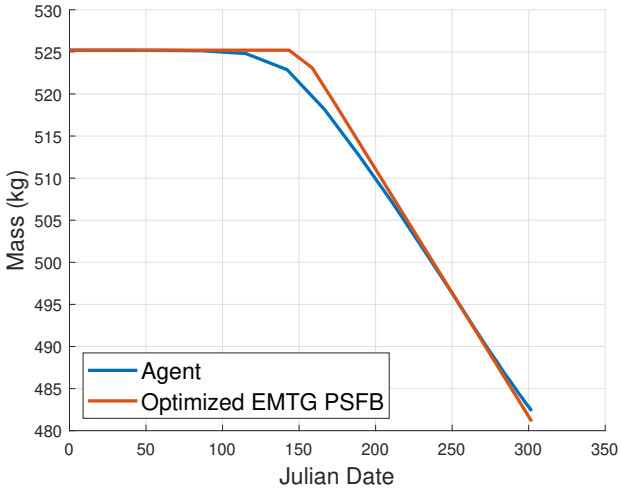

Figure 3-6: Case 2 Mass History

Chapter 4

Cis-lunar Lyapunov Orbit Perturbation Correction

In this chapter, an RL controller will be presented that is capable of transfers between planar Lyapunov orbits to evaluate the potential of such a system for onboard control. Rather than generate its own trajectory between orbits, this agent was tasked with determining the best way to return to a precalculated transfer trajectory from a perturbed initial state and station-keep in the resulting orbit. To achieve this, training was completed on a transfer between L_1 and L_2 libration point Lyapunov orbits on a path that remains distant to the Moon. Its performance was then evaluated using Monte Carlo testing at three different levels of initial state error. Three similar but previously unseen scenarios were then presented to the agent: an L_1 to L_2 transfer featuring a close pass of the Moon, an L_2 to L_1 along a far pass, and an L_2 to L_1 featuring a close pass. These will be used to determine the agent's ability to not only generalize its solution to many variants of its trained problem, but also to unknown and more complex situations.

4.1 Problem Formulation

This section will begin with a explanation of the planar circular restricted three body problem (CR3BP), followed by a description of the two orbits and four reference

transfers. The RL implementation will be presented, including the state and action vectors, the neural network structure, and the reward function.

4.1.1 Dynamics

The planar CR3BP was employed to simulate the motion of the spacecraft in cis-lunar space due to its simplicity and the computational friendliness of its scaling. In this model, it is assumed that the first two bodies P_1 and P_2 are massive compared to a third body of insignificant mass P_3, such that $M_1 > M_2 >> M_3$. In the case of this research, these would be the Earth, the Moon, and the spacecraft, respectively. The motion of all three bodies are calculated about the center of mass of the system, known as the barycenter. Upon the barycenter a rotating reference frame is placed with the x-axis collinear with P_1 and P_2. While from an inertial perspective these bodies orbit the barycenter along circular orbits, within this new reference frame of angular velocity ω, the bodies are fixed in place. The only remaining motion in the problem is that of the spacecraft, P_3. Distance, time, and mass are nondimensionalized according to a characteristic length l^* of the distance between the Earth and the Moon, a characteristic time t^* of the period of the rotating reference frame, and through the use of a mass ratio $\mu = \frac{M_2}{M_1+M_2}$. In conclusion, this creates an idealized model of three-body orbital mechanics in which distances are generally less than 1, the motion of the body of interest relative to the two massive bodies is easily understood, and the equations of motion are simplified by the elimination of mass terms and the universal gravitational constant.

An illustration of the planar CR3BP is provided in Figure 4-1. Notice how the locations of the massive bodies P_1 and P_2 are functions of the mass ratio μ. For pairings such as the Earth and the Moon, the larger body is sufficiently massive to place the barycenter within its radius. In the figure, the position and velocity of the third body in the rotating reference frame is represented by the vector $\mathbf{r} = [x, y, \dot{x}, \dot{y}]$. This vector is propagated forward using the equations of motion in Eq. 4.1. These can be found, along with a full derivation, in many standard orbital mechanics texts, such as Schaub and Junkins [55].

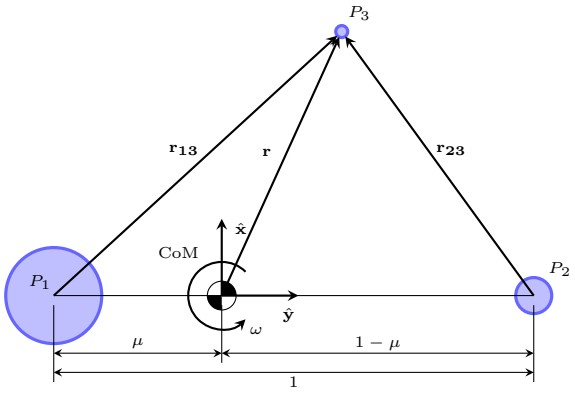

Figure 4-1: Planar CR3BP Illustration recreated from [55]

$$\begin{aligned} \ddot{x} - 2\dot{y} - x &= -\frac{(1-\mu)(x+\mu)}{r_{13}} - \frac{\mu(x-1+\mu)}{r_{23}^3} + a_{\text{lt}}u_x \\ \ddot{y} + 2\cdot -y &= -\frac{(1-\mu)y}{r_{13}^3} - \frac{\mu y}{r_{23}^3} + a_{\text{lt}}u_y \end{aligned} \tag{4.1}$$

These equations are a function of the mass ratio and the distance between the third body and the two fixed bodies. These distances are calculated according to the following equations.

$$\begin{aligned} r_{13} &= \|\mathbf{r_{13}}\| = \sqrt{(x+\mu)^2 + y^2} \\ r_{23} &= \|\mathbf{r_{23}}\| = \sqrt{(x-1+\mu)^2 + y^2} \end{aligned} \tag{4.2}$$

-

The simulated spacecraft utilized a continuously-throttleable, Constant Specific Impulse (CSI) low-thrust engine. The nondimensionalized acceleration provided by this system is represented by the terms $a_{\text{lt}}u_x$ and $a_{\text{lt}}u_y$. Based upon the work of Cox et. al [11], the thrust vector commanded by the agent is given by

$$\mathbf{a}_{\text{lt}} = \frac{f}{m}\hat{\mathbf{u}} = (a_{\text{lt}}u_z)\,\hat{\mathbf{x}} + \left(a_{\text{lt}}u_y\right)\hat{\mathbf{y}} \tag{4.3}$$

where f is the thrust magnitude, m is the nondimensionalized mass of the spacecraft, and $\hat{\mathbf{u}}$ is the directional unit vector the thrust. More specifically, m is the mass ratio

$M_3/M_{3,0}$. Cox et. al suggest values for f between 1.73×10^{-2} nd and 6.95×10^{-2} nd for current spacecraft in the Earth-Moon system, leading to the selection of $\max f = 4 \times 10^{-2}$ nd for this demonstration.

Using these values, as well as the I_{sp} of the spacecraft and the acceleration due to gravity on Earth g_0, the nondimensional mass rate of change can be found. All of the relevant constants to the problem can be found in Table 4.1.

$$\dot{m} = \frac{-fl^*}{I_{sp} g_0 t^*} \tag{4.4}$$

A common value of interest in the CR3BP is Jacobi's integral, C. Originating from the equations of motion, it is the only conserved quantity in the dynamical model and is a measure of the relative energy that a given mass m has.

$$C = 2\left(\frac{1-\mu}{r_{13}} + \frac{\mu}{r_{23}} + \frac{1}{2}\left(x^2 + y^2\right)\right) - \left(\dot{x}^2 + \dot{y}^2\right) \tag{4.5}$$

While slightly unintuitive – larger values correspond to less energy – it provides an important tool when working in the dynamical environment. In this chapter, reference trajectories will utilize heteroclinic transfers between Lyapunov orbits characterized by identical values of C. These are special trajectories that will travel to and asymptotically enter a particular orbit without any thrust. As a result, the spacecraft exchanges potential and kinetic energy along this path and the energy level given by C remains constant. Therefore, it provides a means of evaluating whether a spacecraft is properly following a reference trajectory. Due to the unchanging nature of this scalar in this context, Jacobi's integral is often interchangeably referred to as the Jacobi constant.

μ	l^*	t^*
0.012004715741012 nd	384,747.962856037 km	375,727.551633535 sec

I_{sp}	g_0	$\max f$
3,000 sec	9.80665 m/ sec	0.04 nd

Table 4.1: CR3BP Characteristic Constants and Parameters

4.1.2 Lyapunov Orbit Transfers

The two selected Lyapunov orbits both feature a Jacobi constant of $C = 3.124102$, thus allowing for a heteroclinic connection to be made. These connections were found by forming an initial guess based on manifold intersections on a Poinecaré map and refined using the methods of Haapala and Howell [31][1] The precise L_1 and L_2 orbits selected are shown in Figure 4-2, with their initial conditions in Tables B.5 and B.6.

Four reference trajectories were tested. Reference 1 travels from an orbit about the L_1 to one around the L_2 and remains relatively far away from the Moon. By remaining further from the Moon, the velocity of the spacecraft is slower and the sensitivity of the dynamics is reduced. This is the scenario for which the agent was trained and upon which its performance is initially evaluated. However, for an onboard controller to be useful, it must also be capable of handling many different scenarios. The usefulness of a trained ANN controller would be greatly reduced if numerous agents had to be trained before a mission and stored in a flight computer. The second trajectory problem increases the difficultly with a similar transfer from the L_1 to the L_2 that passes closer the Moon. Reference trajectories 1 and 2 are both shown in Figure 4-3 and their initial conditions may be found in Table B.7.

To further test the generalization capabilities of the agent, the reversed versions of these transfers were also considered. This helped evaluate the directional independence of the agent's solution. That is to say, it showed whether the agent was capable of reducing its error relative to a reference trajectory while travelling in the opposite direction to which it was trained. Due to the symmetry of the dynamical environment, these transfers from the L_2 to the L_1 are flipped in the y-axis compared to References 1 and 2, but are otherwise largely identical. Designated Reference trajectories 3 and 4, they can be found in Figure 4-4 and their initial conditions may be found in Table B.8.

[1]The identification of orbits and their heteroclinic connections was completed by my collaborator Nicholas LaFarge for our paper [36].

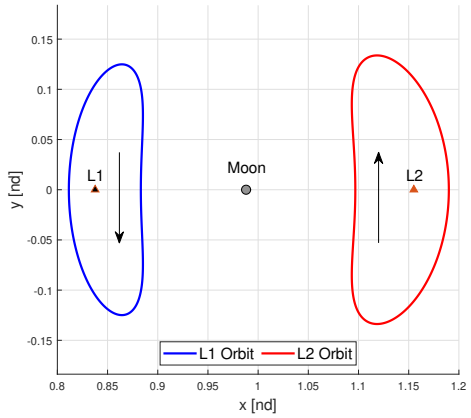

Figure 4-2: L_1 and L_2 Orbits

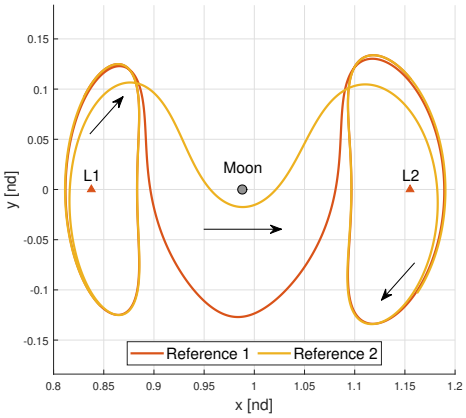

Figure 4-3: Forward Reference Trajectories

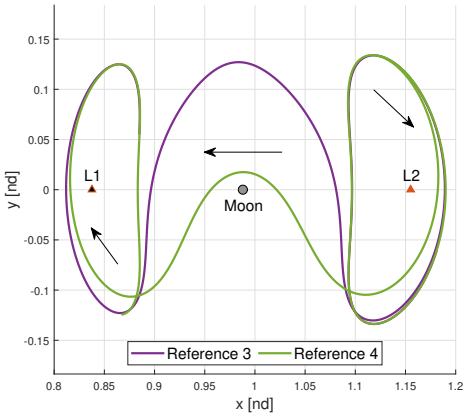

Figure 4-4: Reversed Reference Trajectories

4.1.3 RL Implementation

Each episode was comprised of 100 time steps, unless conditions were met for the early termination of episodes that will be covered in a later section. Time steps were separated by a constant integration time of 0.2 nd, or approximately 20.87 hr. Unlike in the Earth-Mars transfer problem of the previous chapter, a single initial condition was no longer used. Instead, a randomly selected state from the appropriate Lyapunov orbit was selected and an error introduced in position and velocity by sampling normal distributions. A standard deviation of 300 km in position and of $4\,\mathrm{m\,s^{-1}}$ in velocity was used, with a mean of 0 for both.

State Definition

The first priority when designing the state of the agent was to fulfill the Markov property. Recall that this requires the transition from an initial state to a subsequent state to only be a function of the former. In the rotating reference frame of the CR3BP, the position of the Earth and the Moon are fixed, and the dynamics are therefore not a function of time. To fully define the agent's dynamical state, it is consequently only necessary to provide the position, velocity and mass of the spacecraft: $x.$ y, $\dot{x}$, $\dot{y}$, and m.

The agent must also be provided with information regarding its objective. One concern that arose when considering previous implementations such as that of Chapter 3 was the dependence on time. Unlike in the Earth-Mars transfer problem, the arrival time is considered unimportant. Further, any dependence on time may be detrimental to the agent, since the goal of correcting the initial state error should be to consistently return to the desired reference trajectory, not to catch an arbitrary moving point along it. To facilitate this goal, a new method known as the nearest neighbor search was implemented. Instead of calculating the error of the agent's dynamical state with respect to a point along the trajectory as a function of mission elapsed time, the point with the least state error was selected instead. Note that this point could be in the transfer trajectory or the target orbit. Using this new functionality, error components were appended to the agent's state: x_{err}, y_{err}, $\dot{x}_{\text{err}}$, and $\dot{y}_{\text{err}}$.

Two additional observations were provided to the agent. Recalling that the Jacobi constant for both orbits and the path between them is a conserved value, it was decided to provide the agent with both the desired Jacobi constant C_{ref} and its actual value $C_{\mathbf{s}_t}$. By comparing these values, a measure of total error could be determined. With the addition of these final two variables, the agent's state at time step t was defined as

$$\mathbf{s}_t = [x, y, \dot{x}, \dot{y}, m, x_{\text{err}}, y_{\text{err}}, \dot{x}_{\text{err}}, \dot{y}_{\text{err}}, C_{\mathbf{s}_t}, C_{\text{ref}}]^T \tag{4.6}$$

Action Definition

Actions were defined in a similar manner to the controller of Chapter 3. The raw output of the policy ANN was a 3×1 vector containing an unscaled thrust magnitude $\tilde{f}$ and unnormalized directional vector components $\tilde{u}_x$ and $\tilde{u}_y$. All three components were bounded between -1 and 1.

$$\mathbf{a}_t = \begin{bmatrix} \tilde{f} & \tilde{u}_x & \tilde{u}_y \end{bmatrix}^T \tag{4.7}$$

These selections were processed within the environment code into a usable control signal. The thrust was first mapped to a range of 0 to 1 and then multiplied by the maximum thrust of the spacecraft, $f_{\max}$ =0.04 nd. This resulted in a thrust magnitude $f \in [0, f_{\max}]$.

$$f = \frac{\tilde{f} + 1}{2} f_{\max} \in [0, f_{\max}] \tag{4.8}$$

In order to define a direction for the application of thrust, a unit vector was calculated from the second and third components of the action.

$$\hat{\mathbf{u}} = \begin{bmatrix} u_x & u_y \end{bmatrix} = \frac{\begin{bmatrix} u_x & u_y \end{bmatrix}}{\| \tilde{u}_x \quad \tilde{u}_y \|} \tag{4.9}$$

Reward Function and Termination Conditions

The design of the reward function required careful consideration of the desired end product and of any local minima that could derail the learning process. The primary objective of the agent is to eliminate state error with respect to the reference trajectory. By following this trajectory, the agent asymptotically approaches and enters the desired final orbit. Since the agent at all times remains close to the reference, an exponential term $e^{-c_1 \rho_t}$ was chosen as the basis of the reward, where ρ_t is the ℓ_2-norm of the position and velocity error.

A local minimum quickly became apparent within this reward structure. Heteroclinic transfers do not depart their starting orbit suddenly, but rather slowly over multiple revolutions. Rather than depart the region of the initial orbit, the agent

would frequently learn to correct the initial perturbation and station-keep in the initial orbit using this early portion of the transfer as a guide. To correct for this, the exponential term was scaled by a factor λ that increased along the reference trajectory, thus encouraging the agent to continue along the trajectory. The scaling function is defined as

$$\lambda = \frac{\mathfrak{i}}{\mathfrak{n}} c_2 + 1 \tag{4.10}$$

where $\mathfrak{i}$ is the nearest neighbor index along the reference trajectory, $\mathfrak{n}$ is the total number of indices in the reference trajectory, and c_2 is a user selected coefficient. If the nearest neighbor is located in the initial orbit, $\lambda = 1$, while $\lambda = c_2 + 1$ in the arrival orbit.

Penalties and bonuses were also defined for specific circumstances, creating a piecewise reward function. If the agent was determined to have sufficiently reduced the state error such that the position and velocity errors met the criteria $\rho_{t_{\text{pos.}}} < \rho_{t_{\text{pos., arr.}}}$ and $\rho_{t_{\text{vel.}}} < \rho_{t_{\text{vel., arr.}}}$, the reward was set to a value $\mathfrak{b}_{\text{arrival}}$. If the agent diverged from the reference trajectory such that $\rho_{t_{\text{pos.}}} > \rho_{t_{\text{pos., max}}}$ or $\rho_{t_{\text{vel.}}} > \rho_{t_{\text{vel., max}}}$, a deviation penalty of $\mathfrak{p}_{\text{deviate}}$ was applied. In both of these cases, the episode would then immediately terminate. Taken together, the reward function was defined as

$$r_t = \begin{cases} \mathfrak{b}_{\text{arrival}}, & \text{if } \rho_{t_{\text{pos.}}} < \rho_{t_{\text{pos., arr.}}} \text{ or } \rho_{t_{\text{vel.}}} < \rho_{t_{\text{vel., arr.}}} \\ \mathfrak{p}_{\text{deviate}}, & \text{if } \rho_{t_{\text{pos.}}} > \rho_{t_{\text{pos., max}}} \text{ or } \rho_{t_{\text{vel.}}} > \rho_{t_{\text{vel., max}}} \\ \lambda e^{-c_1 \rho_t}, & \text{otherwise} \end{cases} \tag{4.11}$$

4.2 Results

The following analysis of the trained agent's performance will be broken into 4 parts. First, the primary task of the L1 to L2 Reference 1 transfer will be tested. Next, the ability of the agent to generalize its control solution to Reference 2-4 will be evaluated. Both of these sections will make use of results produced by Monte Carlo testing. The limits of the agent will then be shown by providing examples of common

Hyperparameter	Value
Discount Factor γ	0.89
Number of Episodes	170,000
Frequency of Update	1 update per 10 episodes
Batch Size	64
Reward Slope c_1	-500
Trajectory Bonus c_2	1
Arrival Bonus $\mathfrak{b}_{\text{arrival}}$	16.203
Deviation Penalty $\mathfrak{p}_{\text{deviate}}$	-4

Table 4.2: Trained Agent Hyperparameters

failure scenarios. In the final section, the reproducabiliity of the results will be investigated with statistics regarding the performance of 200 agents trained with identical hyperparameters.

Hyperparameters were selected following a grid search of possible values. This necessitated the training of many hundreds of agents and was made possible by leveraging the MIT SuperCloud high performance computing resources [53]. The selected parameters are shown in Table 4.2. To produce the initial state error at the beginning of each episode, errors were randomly applied by sampling normal distributions in position and velocity with standard deviations of 300 km and $4\,\mathrm{m\,s^{-1}}$, respectively.

Three error levels were defined for the purpose of assessing the impact of increasing perturbation levels on the agent's performance. According to Guzzetti et. al, a reasonable estimate of orbit determination errors during cis-lunar station-keeping operations are 1 km in position and $0.01\,\mathrm{m\,s^{-1}}$ in velocity [30]. Based on this, three error levels were defined using normal distributions with 3σ bounds 10x, 100x, and 1000x greater than these values.

4.2.1 L1 to L2 Far Pass

In the task for which it was trained, the agent demonstrated a consistent capability to complete the L_1 to L_2 far pass transfer. To illustrate this, an example transfer with a large perturbation is provided. Note that the episode was not terminated after meeting the arrival condition in order to better illustrate the station-keeping

capabilities of the agent. First, a randomly selected L_1 orbit state was selected and a perturbation of 660.6 km and 7.0 m s^{-1} was applied by sampling the 1000x error distributions. Since the error term in the agent's state is calculated with respect to the nearest point along the heteroclinic transfer, the initial error to correct is slightly different. In this case, it was approximately 116.8 km and 0.5 m s^{-1}.

In Figure 4-5, the agent can be seen to command high thrust levels over the first three time steps in order to acquire the reference trajectory. Note that the each set of 5 arrows of consistent color, length, and direction represent the control being held constant over the integration length of one time step, rather than 5 individual time steps with an unchanging control law. The control history during this transfer is shown in Figure 4-6. The three aforementioned high-thrust arcs are shown during the first 5 days of the transfer, followed by a mostly consistent, sinusoidal thrust pattern. These latter commands correspond to the many revolutions of the L_2 orbit completed by the agent upon arrival. For clarity, actions below the threshold of 25% of $f_{\max}$ were not plotted in Figure 4-5 or in any other figure within this chapter.

The agent's broader performance in the Reference 1 task was evaluated via Monte Carlo testing consisting of 10,000 episodes at each of the 3 error levels starting from randomly selected locations within the L_1 orbit. Shown in Table 4.3, the agent was able to successfully complete the transfer from over 99% of the starting locations at all 3 perturbation levels. While a small decrease in performance with increasing error was measured, the success rate is largely insensitive to the magnitude of the initial state error.

Error Level	Reference 1	Reference 2	Reference 3	Reference 4
10x	99.83%	99.15%	90.61%	99.22%
100x	99.72%	99.07%	89.68%	99.20%
1000x	99.09%	97.49%	87.65%	99.02%

Table 4.3: Monte Carlo Results

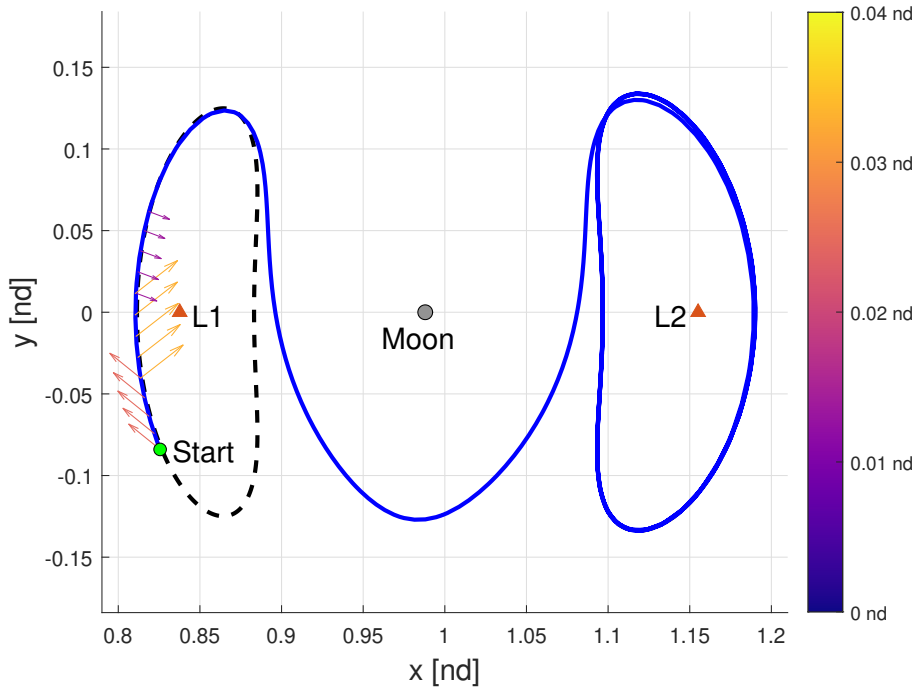

Figure 4-5: Reference 1 at High Error Level - Trajectory

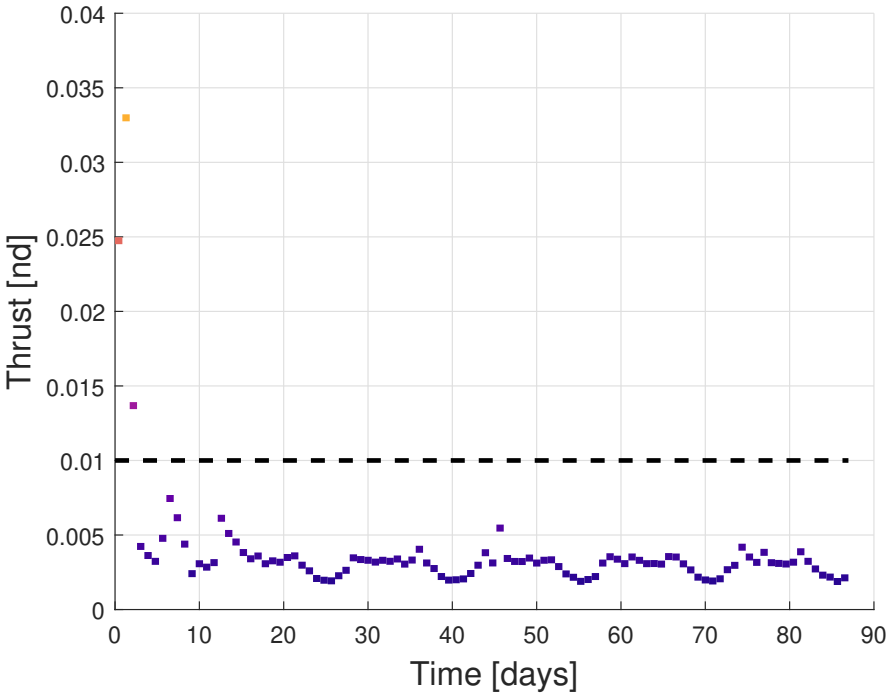

Figure 4-6: Reference 1 at High Error Level - Control History

4.2.2 Generalization

It is important to evaluate the agent's ability to generalize its control solution. Beyond simply being more useful if it can solve other similar problems, this test helps confirm that the agent's policy is not dependent on factors specific to the training problem, such as travelling in the direction of the positive x-axis. Transfers along References 2, 3, and 4 can be found in Figures 4-7 - 4-12. Similar to the Reference 1 scenario, in each case the agent used an initially large thrust level to correct for the error and jump onto the appropriate transfer trajectory. One notable difference can be found in Figures 4-7 and 4-11. Due to their close proximity to the Moon, the dynamics in the close pass references are more sensitive and the gravitational attraction of the Moon is stronger. These factors amplify any state error and necessitate substantial correction maneuvers immediately after passing the Moon.

The results of the Monte Carlo testing of 10,000 episodes per reference and error level are shown in Table 4.3. In Reference 2, performance was nearly identical to that of Reference 1, with a small drop to 97.49% at the highest error levels. The results for References 3 and 4 were surprisingly at odds with the underlying work presented in [36]. It was expected that the agent would perform at a higher level in the test of Reference 3 than of Reference 4, due to the aforementioned dynamics. This was the behavior seen in previous work. However this agent's performance on Reference 4 was second only to that on Reference 1. In contrast, Reference 3 showed a decrease in performance by approximately 10% to the other tests, though still 8% higher than in [36].

While the agent demonstrated a consistent capability to complete the reverse transfers of References 3 and 4, the thrust profiles in Figures 4-10 and 4-12 show that higher thrust levels were required to complete the station-keeping operations about the L_1. This may suggest that the agent has not mastered these transfers to the same degree as their L_1 to L_2 counterparts. However, the near perfect score during Monte Carlo testing of Reference 4 would suggest that this was not the case. In Figures 4-9 and 4-11, the station-keeping maneuvers with the largest thrust magnitude

were roughly aligned with the x-axis and in the opposite direction of the dominant gravitational body, depending on the location of the spacecraft. Based upon the eigenvalues of the linearized dynamics at the libration points (see Appendix C.1), the motion in the x-axis at the L_1 libration point is more unstable than at the L_2. This orbit is, however, not at the L_1, but rather about it. Further analysis would be necessary to draw a more definitive conclusion, though this too would be difficult due to the black box nature of ANNs.

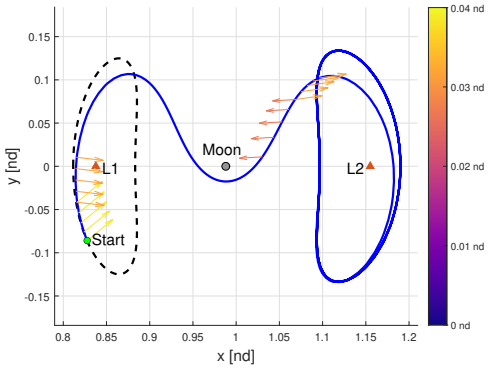

Figure 4-7: Reference 2 at High Error Level - Trajectory

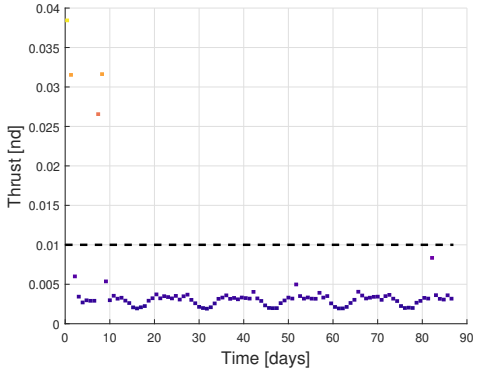

Figure 4-8: Reference 2 at High Error Level - Control History

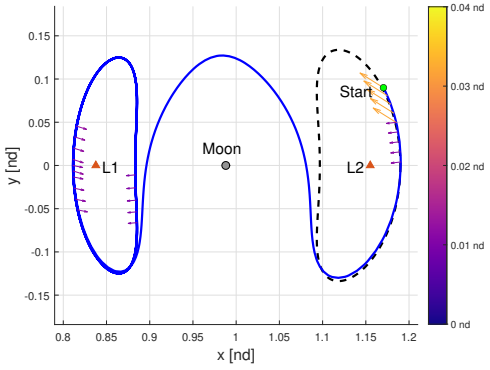

Figure 4-9: Reference 3 at High Error Level - Trajectory

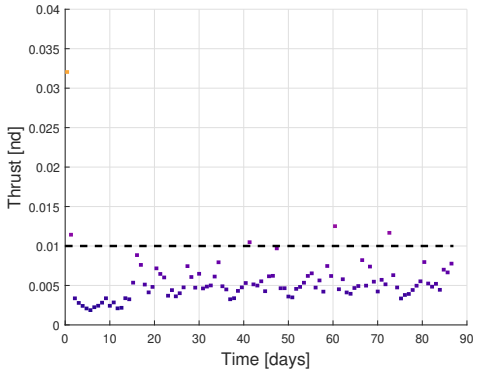

Figure 4-10: Reference 3 at High Error Level - Control History

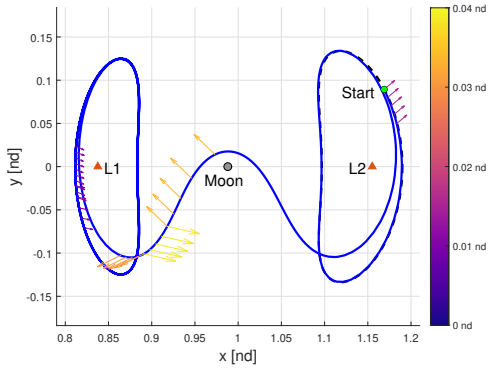

Figure 4-11: Reference 4 at High Error Level - Trajectory

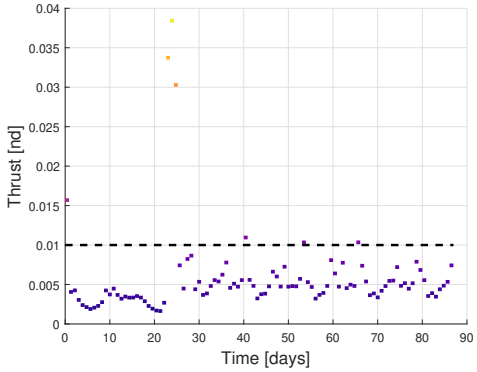

Figure 4-12: Reference 4 at High Error Level - Control History

4.2.3 Failure Modes

Three common failure modes became noticeable during Monte Carlo testing that are worth discussing. The most common failure mode occurred when the agent successfully tracked the reference trajectory, but failed to reduce the error sufficiently to trigger the arrival condition that ends the episode. Such a trajectory is shown in Figure 4-13. The path in green shows that, should the initial perturbation have remained uncorrected, the spacecraft would have collided with the Moon. The agent avoided this fate and appeared to enter the L_1, but without satisfying the required accuracy. It is possible that the error would have been driven to zero with more time, but this was not achieved within the 100 time steps provided.

A less common cause of failure occurred when the agent began near the intersection of the initial orbit and the reference trajectory departing that orbit. Shown in Figure 4-14, the agent needed to decide between immediately progressing with the transfer or going around the libration point. As shown by the uncontrolled solution in green, it would have required minimal thrust to take the latter option. However, the agent decided to proceed towards the L_2, and even with the maximum allowable thrust, it was unable to join the transfer trajectory. As a result, the divergence end condition was triggered.

A final failure mode is that of the unrecoverable initial perturbation. Mostly a

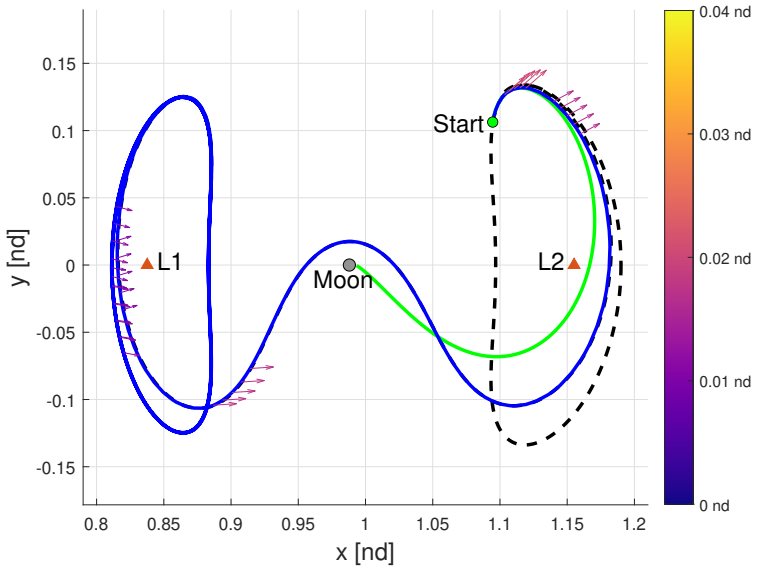

Figure 4-13: Failure due to Insufficient Error Reduction for Arrival

concern at the highest initial error levels, it is occasionally possible that one of the tested perturbations exceeded the capability of the agent to correct, even at maximum thrust. Such a scenario can be found in Figure 4-15. The uncontrolled trajectory can be seen to diverge back towards the Earth and the agent continuously thrusts in the opposite direction to prevent this. However, it was unsuccessful and eventually the divergence end condition was triggered.

This list of failures should not be considered exhaustive, but does highlight some of the limitations of this RL method. Sensitivity to small errors in the first failure case is similar to accuracy issues discussed in Chapter 3, though is clearly less of an issue in this application due to the scaling of the CR3BP environment. The second failure mode shows that the intelligence of the agent is not complete. Further training, perhaps with a greater emphasis on the ambiguous start locations, may address the problem, as would a mixed reinforcement and supervised learning approach that demonstrates the solution to the agent. The third failure mode is arguably not a failure of the agent, unless a portion of these final failures were edge cases solvable by an optimal control method. On that final topic, the optimality of the actions taken by

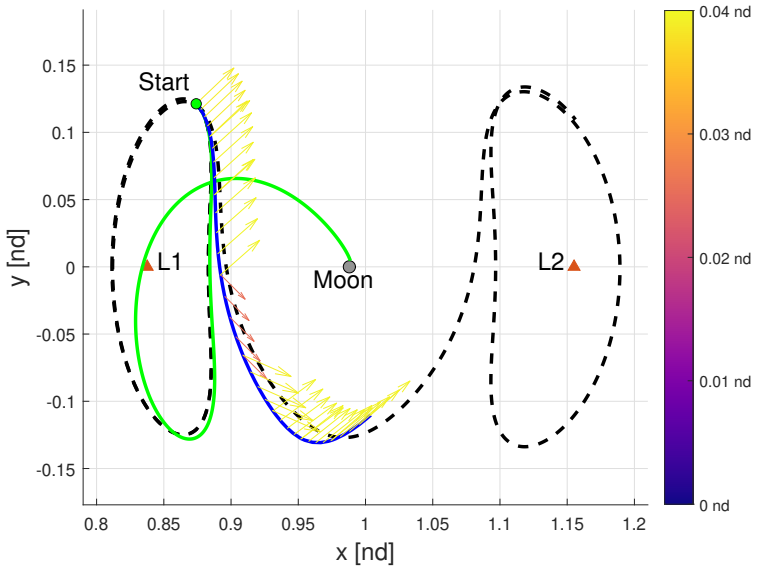

Figure 4-14: Failure due to Ambiguous Thrust Direction

the agent were not evaluated within this investigation and is an area worth studying in the future.

4.2.4 Reproducibility

The most significant problem faced during the development of the PPO-trained RL controller in the CR3BP was reproducability. On average, only 15%-20% of agents exhibited an acceptable level (90%+) of performance in the Reference 1 training problem during Monte Carlo testing. The capability to consistently complete all four transfers would frequently occur in only around 1% of agents at the lower error levels, and an agent capable of achieving this at the highest levels of error has thus far proved elusive. To illustrate this issue, 200 agents were trained – including the one used in this chapter – using identical hyperparmeters. Monte Carlo testing consisting of 10,000 episodes at each error level and on each reference was conducted for each agent. The statistics of these runs can be found in Table 4.4. The average agent was far less successful than the demonstrated agent on the previous sections, scoring between 52% and 60% on the trained reference trajectory and far lower in the remaining three

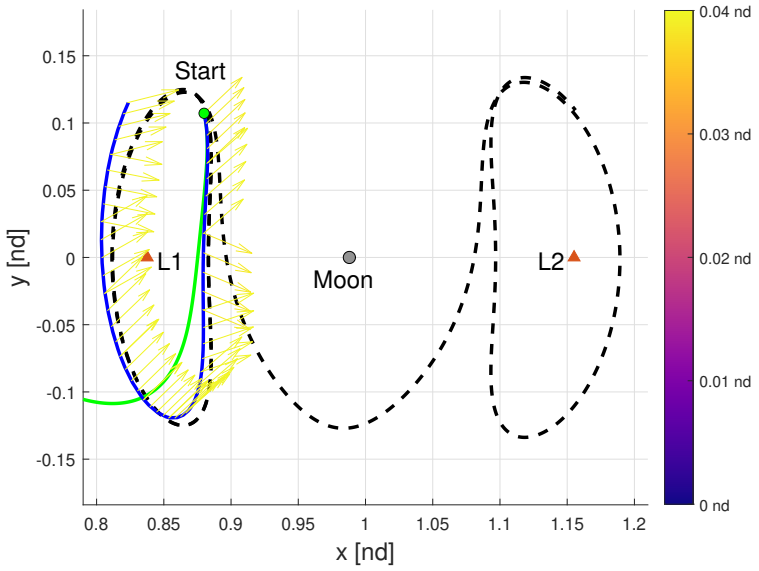

Figure 4-15: Failure due to Unrecoverable Initial Perturbation

transfers. However, it is also clear that the standard deviations of this performance are quite large, indicating a wide spread in performance. The mean and standard deviations of References 3 and 4 were notably lower than those of Reference 1 and 2, indicating that generalizing to the reverse transfers occurred much less frequently.

The statistics do not fully explain the situation, however. If the success is defined as completing transfers at a rate of 90% or higher, then an agent with a score of 80% is no more welcome than an agent with a score of 20%. To better visualize the rate

Error Level	Reference 1		Reference 2	
	Mean	Std. Dev.	Mean	Std. Dev.
Low	59.09%	32.76%	35.03%	33.41%
Medium	58.56%	32.58%	34.96%	33.46%
High	51.33%	31.48%	32.53%	33.46%

Error Level	Reference 3		Reference 4	
	Mean	Std. Dev.	Mean	Std. Dev.
Low	15.76%	24.01%	16.72%	23.80%
Medium	15.82%	23.96%	16.54%	23.74%
High	14.77%	22.98%	15.23%	23.22%

Table 4.4: Average Performance across 200 Agents

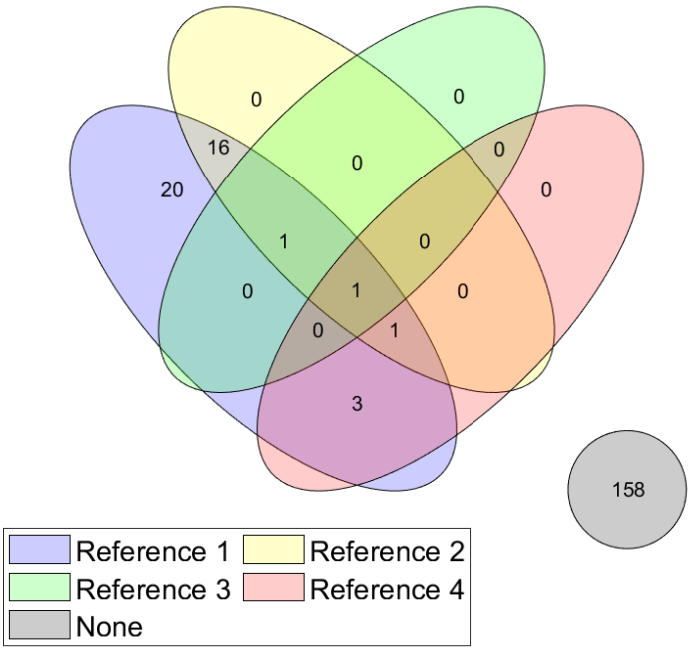

Figure 4-16: Monte Carlo Results of 10,000 Agents at Low Error

at which agents achieved this high standard, a four-set venn diagram was created for each of the three error levels. These are shown in Figures 4-16, 4-17, and 4-18. In each of the ovular regions, the number of agents that successfully completed the associated transfer is labelled, with the colored regions containing agents that were capable of more than one reference transfer. In Figure 4-16, for example, 20 agents completed Reference 1 and only Reference 1 at the desired level, while 3 agents could complete both References 1 and 4. Unsurprisingly, no agents exists outside of the blue oval – that would be an agent that failed the test for the trained transfer but passed for the untrained ones. Across all three error levels, the number of agents that could complete both References 1 and 2 did not lag far behind the number that can complete only Reference 1. This matches the higher mean success rates shown in the previous table. Performance in References 3 and 4 was lower, as expected.

From these diagrams, the challenge quickly becomes clear. What is desired is an agent that exists in the small, overlapping region at the center of the venn diagram, and that does so in all three figures. The vast majority of agents, fail to succeed at

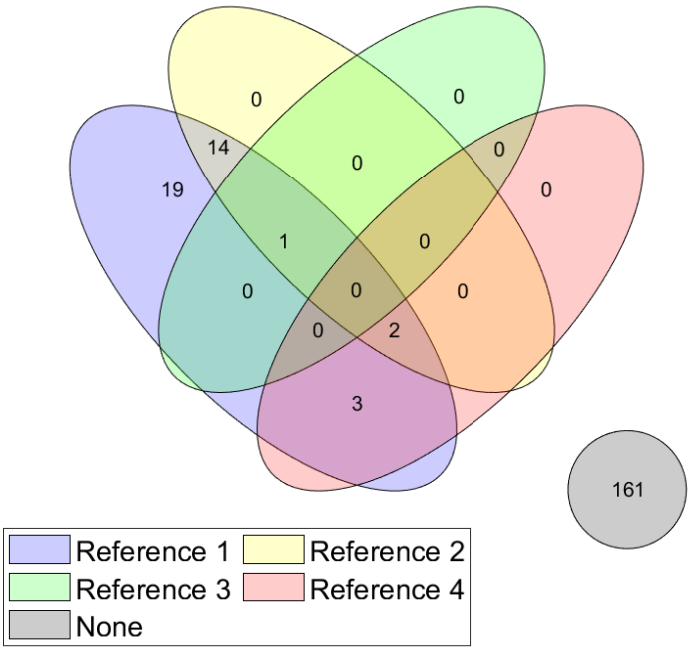

Figure 4-17: Monte Carlo Results of 10,000 at Medium Error

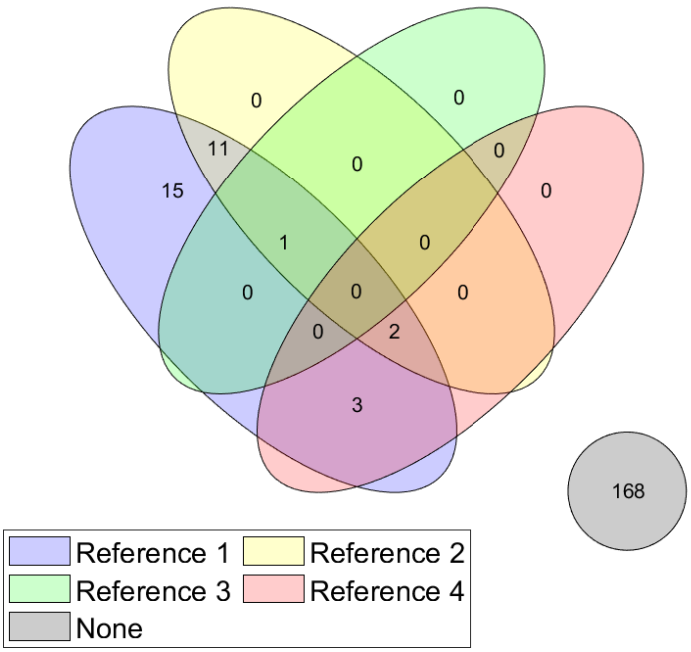

Figure 4-18: Monte Carlo Results of 10,000 at High Error

any category, as shown by the more than 150 agents in the gray region to the right of each figure. The agent used to produce the results in the previous sections, capable of all four references and the solitary 1 at the center of Figure 4-16, proved unique among the 200 agents.

Chapter 5

Conclusion

Machine learning using neural networks is a rapidly progressing technology that promises to revolutionize many fields. In particular, reinforcement learning agents have been demonstrated that are capable of solving complex tasks at a level beyond that of a human. In this thesis, the RL algorithm known as Proximal Policy Optimization was applied to low-thrust spacecraft guidance and control problems to better understand its applicability to such tasks. More specifically, this investigation endeavored to answer two questions. The first was whether PPO could be used as an optimization tool in lieu of optimal control software for purpose of trajectory design. Current optimizers are computationally expensive and struggle with local minima. Although they are expensive to train, RL agents may be better able to find unique trajectory solutions, while also avoiding local minima. The second question was whether an agent trained using PPO would serve as a controller for low-thrust spacecraft in complex dynamical regimes. Once trained, they can provide a low-cost control solution well suited to the limited computational power of the radiation-hardened CPUs used on spacecraft. Furthermore, they can help fulfill requirements for increased autonomy in future spacecraft.

To answer the first question, an agent trained using the clipped objective function variant of the PPO algorithm was tasked with completing a 302 d fixed-length transfer between the Earth and Mars. While broadly successful, it lacked the accuracy of the optimal control solution provided by EMTG. The RL framework also required

initial conditions to be provided to it. While optimizers require an initial guess, they are able to then search for a better initial state. The initial state in RL is, in contrast, uncontrollable by the agent. The RL agent was not faster to produce than the EMTG either. Finally, tweaking a reward function, such that penalty-method constraints could be met, proved challenging. These factors collectively make RL a poor replacement for current trajectory design techniques.

The second question was addressed by training an agent as a regulator controller for a transfer between Lyapunov orbits in cis-lunar space. Trained using the KL divergence version of PPO, the agent was tasked with correcting for an initial state error, following a precalculated reference trajectory, and completing stationkeeping operations. In the L_1 to L_2 task upon which it trained, the agent had a success rate of over 99% at three different levels of initial error in Monte Carlo testing of 10,000 episodes each. It's ability to generalize its control solution and solve three similar transfers was then examined. The agent achieved near parity with its performance in the trained transfer in two of these tasks, while achieving lesser but still impressive results in the remaining transfer. Unlike the Earth to Mars transfer, the accuracy achieved in these tasks was acceptable. This is partially because the priority for onboard systems is to provide feasible solutions; optimality may not be required. It is also due to the scaling of the problem. Cis-lunar space is a smaller environment than the great expanse between the Sun and Mars. Even though that agent's performance in both tasks is similar in nondimensionalized units, the smaller characteristic length in this problem results in far smaller state errors. These combination of factors suggest that RL is a promising method for achieving onboard autonomous control in environments with appropriate scaling.

5.1 Future Work

While the results of this research demonstrate the promise of a reinforcement learning approach to onboard control, it is only an initial application of the technology. More work will be required to address its present limitations. The lack of consistent

reproducability of the results in the CR3BP application of Chapter 4 was a significant challenge that needs to be addressed. Interestingly, this was not an issue with the Earth-Mars transfer of Chapter 3. For the CR3BP agent, the KL divergence version of PPO was used, and training data was collected into batches. In contrast, the research presented in Chapter 3 made use of the clipping variant of the PPO algorithm and did not batch episode data. Each episode also started from the same state and lacked any of the random perturbations applied in Chapter 4. Identifying which, if any, factors may be linked to more consistent results or superior generalization would be a beneficial focus of future research.

The difficulty of implementing constraints during the training of an RL agent is a limitation that is worth addressing. This is, unfortunately, a larger issue within the reinforcement learning framework and not one specific to the problems considered in this thesis. One possible solution may be to calculate the optimal penalty coefficients using methods developed for solving lagrange multipliers in optimal control theory. Without a solution to this problem, the usefulness of an RL controller may remain limited to simpler problems. In a practical implementation, an outer loop outside of RL may be required to better optimize the RL solution based on constraints that could not be included within the reward function. Even though it is less of a concern for onboard systems, the optimality of the RL solution should still be studied in the future to better quantify the additional fuel or time that may be required when using an RL controller as opposed to a more well-established method.

Finally, while pure reinforcement learning offers the theoretical advantage of a complete lack of bias towards known solutions, the use of a combined supervised and reinforcement learning approach may be more practical. The ability to create novel solutions is not as beneficial for an onboard controller and this alternative approach may help eliminate known failure modes such as the ambiguous thrust direction and help improve the controller's optimality.

Appendix A

Nomenclature

A	Action random variable
$\mathbf{a}$	An action taken by the agent
$\mathcal{A}$	Set of all possible actions
κ	A constant value used in Adagrad to prevent a possible divide by zero scenario
g	Gradient. This notation is used in Adagrad, Adam, and PPO
G	Gradient matrix in the Adagrad algorithm
β	Adam decay rate
m	First moment (mean) of the gradient in Adam
$\hat{m}$	Bias-corrected first moment (mean) of the gradient in Adam
ϵ	A constant used in Adam to prevent a divide by zero scenario
F	Adam objective function
v	Second moment (uncentered variance) of the gradient in Adam
$\hat{v}$	Bias-correct second moment (variance) of the gradient in Adam
α	Adam gradient step size

A	Advantage function or value
a_{lt}	Nondimensionalized thrust adjusted for spacecraft mass
n	Gradient descent batch size
ϵ	Clipping limit in PPO
$\mathfrak{T}$	Integration time commanded by the agent
$\tilde{\mathfrak{T}}$	The integration time as produced by the actor's policy before mapping
T	Probability density function for a continuous state transition
γ	Discount factor
T	Final time step in an RL episode
η	Gradient step size
u	Gradient update step index
$\mathfrak{i}$	The index along the reference trajectory selected by nearest neighbor search
C	Jacobi Constant
C_{ref}	The Jacobi integral value of the reference trajectory
$C_{\mathbf{s}_t}$	The Jacobi integral value at timestep t
KL	KL divergence operator
d	KL divergence value
ζ	Limit on the KL divergence
β	PPO KL penalty coefficient
$\mathrm{d}_{\mathrm{targ}}$	Target KL divergence value
$\mathfrak{n}$	The length (number of indices) of the reference trajectory

L_1	The first Earth-Moon Lagrange point, located between the Earth and the Moon
L_{sc}	Earth-Mars Length Scaling Factor
l^*	Characteristic length in the CR3BP used for nondimensionalization
L_2	The second Earth-Moon Lagrange point, located past the Moon along the x-axis of the CR3BP Rotating Frame
m	Spacecraft mass
M_3	CR3BP third body (spacecraft) mass with units
$M_{3,0}$	Initial CR3BP third body (spacecraft) mass with units
$\dot{m}$	Rate of change of spacecraft mass
μ	Mass ratio of the CR3BP
v	Weighted sum of the gradient using a momentum-based gradient descent method (includes Adam)
ζ	Weighting hyperparemeter in momentum gradient descent
M_1	Mass of the first massive body P_1 in the CR3BP
M_2	Mass of the second massive body P_2 in the CR3BP
μ_{sun}	Gravitational coefficient for two-body orbital mechanics about the Sun
σ	Activation function for a neural network layer
o	Output from a given layer
b	Bias matrix of a neural network layer
L	Index for the final layer of a neural network
K	Final neuron index for a given layer (see 2-1)
l	Index to refer to the lth layer of a neural network

j	Index to refer to the jth neuron in a layer
k	Index to refer to the kth neuron in a layer
θ	Parameters (the weights and biases) of a neural network
w	Weight matrix of a neural network layer
J	Objective function
$\mathbf{o}$	Observation
π^*	Optimal policy
V^*	Optimal value function
d	Number of optimization parameters in Adagrad
p	Pre-activation function output from a given layer
h	Parameter index
θ_{old}	The old set of ANN policy parameters
π	Policy function
P_1	First body in the CR3BP
$\rho_{t_{\text{pos.}}}$	Position error at time step t
$\rho_{t_{\text{pos., max}}}$	The maximum permissible state error
R	PPO probability ratio
P_3	Third body in the CR3BP
P_2	Second body in the CR3BP
r	Distance between bodies. Here, it is from the Sun to the spacecraft
r_{13}	Distance from the Earth to the spacecraft in the CR3BP (bolded for vector)

R Reward random variable

r A reward

λ CR3BP reward function scaling parameter along reference trajectory

$\mathcal{R}$ Set of all possible rewards

S State random variable

t Time step in an RL episode

V (State) Value function

$\mathfrak{V}$ Actual value of a state as calculated at the end of an episode

r_{23} Distance from the Moon to the spacecraft in the CR3BP (bolded for vector)

$\mathfrak{p}_{\text{spatial}}$ Penalty for exceeding the allowable environmental spatial bounds

$\mathbf{s}$ A state

ρ_t The state error at time step t relative to the reference

$\mathbf{s}'$ Next state, equivalent to $\mathbf{s}_{t+1}$

$\mathcal{S}'$ Set of all possible next states

$\mathcal{S}$ Set of all possible states

Q State-action value function

t_{elapsed} Nondimensionalized and scaled mission elapsed time with $302\,\text{days} = 1$

f Thrust magnitude commanded by agent

$\tilde{f}$ Raw thrust output of the agent before mapping

$\tilde{u}$ Raw thrust vector components from actor before normalization. A subscript of x, y, or z indicated direction

$\mathfrak{p}_{\text{time}}$ Penalty for exceeding the allowable mission time

ξ Neural network input

i Index of training data in gradient descent

ψ Desired neural network output

n Data set size used in backpropagation

p State transition probability

T_{sc} Earth-Mars Time Scaling Factor

t_{scaled} Time step counter scaled to between 0 and 1

t^* Characteristic time in the CR3BP used for nondimensionalization

u Thrust vector components with subscript of x, y, or z to indicate direction

$\hat{\mathbf{u}}$ Thrust direction unit vector

$\rho_{t_{\text{vel.}}}$ Velocity error at time step t

$\rho_{t_{\text{vel., max}}}$ The maximum permissible velocity error

V_{sc} Earth-Mars Velocity Scaling Factor

ω Constant angular velocity of the rotating CR3BP reference frame

x Distance in the x-axis

$\ddot{x}$ Acceleration in the x-axis

$\dot{x}$ Velocity in the x-axis

$\hat{\mathbf{x}}$ CR3BP x-axis unit vector

y Distance in the y-axis

$\ddot{y}$ Acceleration in the y-axis

$\dot{y}$ Velocity in the y-axis

$\hat{\mathbf{y}}$ CR3BP y-axis unit vector

z Distance in the z-axis

$\ddot{z}$ Acceleration in the z-axis

$\dot{z}$ Velocity in the z-axis

Appendix B

Tables

B.1 Mars-Earth Transfer

Variable	Dimensionalized	Nondimensionalized
x	130,325,632,242.48067 km	0.8711730430419905
y	66,016,869,369.94133 km	0.4412955148693369
z	28,620,014,942.33610 km	0.19131298333418004
$\dot{x}$	$-11{,}891.56920\,\mathrm{km\,s^{-1}}$	-0.3992510403276126
$\dot{y}$	$27{,}870.09774\,\mathrm{km\,s^{-1}}$	0.9357188550630681
$\dot{z}$	$13{,}484.93482\,\mathrm{km\,s^{-1}}$	0.45274716608763854
m	525.2 kg	1

Table B.1: Earth-Mars Transfer Initial Conditions

Variable	Dimensionalized	Nondimensionalized
$\max \mathfrak{T}$	2,544,047.999997586 s	0.5065158035822943
$\min \mathfrak{T}$	127,202.3999998793 s	0.0253257901791147

Table B.2: Earth-Mars Transfer Additional Parameters

Episodes (thousands)	Actor	Critic
0-25	0.0001	0.0005
25-45	0.00005	0.00025
45-60	0.000025	0.000125
60-100	0.00001	0.00005

Table B.3: Case 1 Learning Rate Schedule

Episodes (thousands)	Actor	Critic
0-25	0.0001	0.0005
25-35	0.00005	0.00025
35-45	0.000025	0.000125
45-75	0.00001	0.00005
75-125	0.000005	0.000025

Table B.4: Case 2 Learning Rate Schedule

B.2 Cis-lunar Transfers

x	y	$\dot{x}$	$\dot{y}$
0.8114469487016518	0	0	0.2645398729614783

Table B.5: Initial Conditions for L_1 Orbit with Period 2.971513438364553 nd

x	y	$\dot{x}$	$\dot{y}$
1.1899997915386646	0	0	−0.23402179666560755

Table B.6: Initial Conditions for L_2 Orbit with Period 3.489271251966925 nd

Variable	Reference 1	Reference 2
x	0.8301451575056924	0.8466786651620697
y	0.09182926530660901	−0.11599449173330423
$\dot{x}$	0.08476444027423845	−0.09631990807174416
$\dot{y}$	0.17406522929410265	0.09347843166919054

Table B.7: Nondimensionalized Initial Conditions for References 1 and 2 with integration time 10 nd

Variable	Reference 1	Reference 2
x	1.1810178660688302	1.12202973104477
y	−0.0606618168212088	0.1335020875801761
$\dot{x}$	−0.06435732588832516	0.09937372384008122
$\dot{y}$	−0.20749958740253738	−0.014301895471061605

Table B.8: Nondimensionalized Initial Conditions for References 3 and 4 with integration time 9 nd

Appendix C

Additional Analysis

C.1 Libration Point Stability

In the analysis of the stability of nonlinear systems about a small region, it is useful to consider the eigenvalues of the equivalent linearized systems. According to Schaub and Junkins [55], the eigenvalues for motion solely along the x-axis λ_x are given by

$$\lambda_x = \pm\sqrt{2E\left(\mathbf{r}_0\right)+1} \tag{C.1}$$

where $\mathbf{r}_0 = [x_0, 0, 0]^T$ is the vector to the position along the x-axis x_0 that corresponds to either the L_1, L_2, or L_3 libration point. This is also the point about which the dynamics are linearized. The constant positive scalar $E\left(\mathbf{r}_0\right)$ is calculated to be

$$E\left(\mathbf{r}_0\right) = \frac{1-\mu}{\|x_0+\mu\|^3} + \frac{\mu}{\|x_0-1+\mu\|^3} \tag{C.2}$$

Using $x_0 = 0.83763530136$ for the L_1, the eigenvalue for motion along the x-axis at the L_1 is found to be approximately $\lambda_x = \pm\ 3.3593$. Using $x_0 = 1.15511844446$ for the L_2, an eigenvalue of $\lambda_x = \pm\ 2.7178$. Based upon the larger magnitude of the L_1 eigenvalue, it can be concluded that motion along the $x - axis$ near the L_1 is more unstable than near the L_2.

Bibliography

[1] Charles Acton, Nathaniel Bachman, Boris Semenov, and Edward Wright. A look towards the future in the handling of space science mission geometry. *Planetary and Space Science*, 150:9–12, jan 2018.

[2] Charles H. Acton. Ancillary data services of NASA's Navigation and Ancillary Information Facility. *Planetary and Space Science*, 44(1):65–70, jan 1996.

[3] Andrew M. Annex, Ben Pearson, Benoît Seignovert, Brian T. Carcich, Helge Eichhorn, Jesse A. Mapel, Johan L. Freiherr Von Forstner, Jonathan McAuliffe, Jorge Diaz Del Rio, Kristin L. Berry, K.-Michael Aye, Marcel Stefko, Miguel De Val-Borro, Shankar Kulumani, and Shin-Ya Murakami. SpiceyPy: a Pythonic Wrapper for the SPICE Toolkit, February 2020.

[4] Jonathan AZIZ, Daniel SCHEERES, Jeffrey PARKER, and Jacob ENGLANDER. A smoothed eclipse model for solar electric propulsion trajectory optimization. *TRANSACTIONS OF THE JAPAN SOCIETY FOR AERONAUTICAL AND SPACE SCIENCES, AEROSPACE TECHNOLOGY JAPAN*, 17(2):181–188, 2019.

[5] Jonathan D. Aziz, Jeffrey S. Parker, Daniel J. Scheeres, and Jacob A. Englander. Low-Thrust Many-Revolution Trajectory Optimization via Differential Dynamic Programming and a Sundman Transformation. *The Journal of the Astronautical Sciences*, 65(2):205–228, jan 2018.

[6] Jonathan D. Aziz, Daniel Scheeres, and Gregory Lantoine. Differential Dynamic Programming in the Three-Body Problem. In *2018 Space Flight Mechanics Meeting*. American Institute of Aeronautics and Astronautics, jan 2018.

[7] Ryne T. Beeson, Jacob A. Englander, Steven P. Hughes, and Maximillian Schadegg. An Automatic Medium to High Fidelity Low-Thrust Global Trajectory Toolchain; EMTG-GMAT. In *25th AAS/AIAA Space Flight Mechanics Meeting*, January 2015.

[8] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives. *CoRR*, abs/1206.5538, 2012.

[9] John T. Betts. Survey of Numerical Methods for Trajectory Optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, March 1998.

[10] Xiaoyu Chu, Kyle T. Alfriend, Jingrui Zhang, and Yao Zhang. Q-Learning Algorithm for Path-Planning to Maneuver through a Satellite Cluster. In *AAS/AIAA Astrodynamics Specialist Conference, Snowbird, UT*. American Astronautical Society, Univelt, August 2018.

[11] Andrew D. Cox, Kathleen C. Howell, and David C. Folta. Dynamical structures in a low-thrust, multi-body model with applications to trajectory design. *Celestial Mechanics and Dynamical Astronomy*, 131(3):12, mar 2019.

[12] Bernd Dachwald. Evolutionary Neurocontrol: A Smart Method for Global Optimization of Low-Thrust Trajectories. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, page 5405. American Institute of Aeronautics and Astronautics, aug 2004.

[13] A Das-Stuart, KC Howell, and D Folta. Rapid Trajectory Design in Complex Environments Enabled Via Supervised and Reinforcement Learning Strategies. In *69th International Astronautical Congress, Bremen, Germany*, 2018.

[14] A Das-Stuart, KC Howell, and DC Folta. A Rapid Trajectory Design Strategy for Complex Environments Leveraging Attainable Regions and Low-Thrust Capabilities. In *68th International Astronautical Congress, Adelaide, Australia*, 2017.

[15] 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, 2019.

[16] Theresa Debban, T. McConaghy, and James Longuski. Design and Optimization of Low-Thrust Gravity-Assist Trajectories to Selected Planets. *American Institute of Aeronautics and Astronautics*, aug 2002.

[17] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[18] Donald H. Ellison, Bruce A. Conway, Jacob A. Englander, and Martin T. Ozimek. Analytic Gradient Computation for Bounded-Impulse Trajectory Models Using Two-Sided Shooting. *Journal of Guidance, Control, and Dynamics*, 41(7):1449–1462, jul 2018.

[19] Donald H. Ellison, Bruce A. Conway, Jacob A. Englander, and Martin T. Ozimek. Application and Analysis of Bounded-Impulse Trajectory Models with Analytic Gradients. *Journal of Guidance, Control, and Dynamics*, 41(8):1700–1714, aug 2018.

[20] Donald Hamilton Ellison. *Robust Preliminary Design for Multiple Gravity Assist Spacecraft Trajectories*. PhD thesis, University of Illinois at Urbana-Champaign, 2018.

[21] Jacob A. Englander and Bruce A. Conway. Automated Solution of the Low-Thrust Interplanetary Trajectory Problem. *Journal of Guidance, Control, and Dynamics*, 40(1):15–27, jan 2017.

[22] B. Farley and W. Clark. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*, 4(4):76–84, sep 1954.

[23] Roberto Furfaro, Ilaria Bloise, Marcello Orlandelli, Pierluigi Di Lizia, Francesco Topputo, and Richard Linares. Deep Learning for Autonomous Lunar Landing. In Brandon A. Jones, Puneet Singla, Ryan M. Weisman, and Belinda G. Marchand, editors, *AAS/AIAA Astrodynamics Specialist Conference, 2018*, Advances in the Astronautical Sciences, pages 3285–3306. Univelt Inc., January 2018. AAS/AIAA Astrodynamics Specialist Conference, 2018 ; Conference date: 19-08-2018 Through 23-08-2018.

[24] Brian Gaudet, Richard Linares, and Roberto Furfaro. Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing. *CoRR*, abs/1810.08719, 2018.

[25] Brian Gaudet, Richard Linares, and Roberto Furfaro. Six degree-of-freedom body-fixed hovering over unmapped asteroids via LIDAR altimetry and reinforcement meta-learning. *Acta Astronautica*, 172:90–99, jul 2020.

[26] Brian Gaudet, Richard Linares, and Roberto Furfaro. Six Degree-of-Freedom Hovering over an Asteroid with Unknown Environmental Dynamics via Reinforcement Learning. In *AIAA Scitech 2020 Forum*. American Institute of Aeronautics and Astronautics, jan 2020.

[27] 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, June 2020.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[29] Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, and Robert Babuska. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, nov 2012.

[30] 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 1–20. AAS Marriott Plaza, Texas, 2017.

[31] 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 Bifurcation and Chaos*, 26(05):1630013, may 2016.

[32] 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*. American Institute of Aeronautics and Astronautics, aug 2008.

[33] Bettina Inclán, Gina Anderson, and Jimi Russell. NASA Awards Artemis Contract for Lunar Gateway Power, Propulsion. Electronically, May 2019.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[35] Donald E Kirk. *Optimal Control Theory: An Introduction*. Courier Corporation, 2004.

[36] 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 *AIAA Scitech 2020 Forum*, page 0458. American Institute of Aeronautics and Astronautics, jan 2020.

[37] Erwan Lecarpentier and Emmanuel Rachelson. Non-Stationary Markov Decision Processes, a Worst-Case Approach using Model-Based Reinforcement Learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 7216–7225. Curran Associates, Inc., 2019.

[38] 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, may 2010.

[39] T Troy McConaghy. GALLOP Version 4.5 User's Guide. Electronically, 2005.

[40] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, dec 1943.

[41] Daniel Miller, Jacob A Englander, and Richard Linares. Interplanetary Low-Thrust Design Using Proximal Policy Optimization. In *2019 AAS/AIAA Astrodynamics Specialist Conference*, August 2019.

[42] Daniel Miller and Richard Linares. Low-Thrust Optimal Control via Reinforcement Learning. In *29th AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI*. American Astronautical Society, Univelt, January 2019. AAS 19-560.

[43] 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(7540):529–533, feb 2015.

[44] Michael A. Nielsen. Neural Networks and Deep Learning, 2018.

[45] Cesar Ocampo and Juan Senent. The Design and Development of COPERNICUS: A Comprehensive Trajectory Design and Optimization System. In *57th International Astronautical Congress*. American Institute of Aeronautics and Astronautics, oct 2006.

[46] NL Parrish and DJ Scheeres. Optimal low-thrust trajectory correction with neural networks. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 1–20, 2018.

[47] Peter Y Peterson, Daniel A Herman, Hani Kamhawi, Jason D Frieman, Wensheng Huang, Tim Verhey, Dragos Dinca, Kristen Boomer, Luis Pinero, Kenneth Criswell, et al. Overview of NASA's Solar Electric Propulsion Project. In *36th International Electric Propulsion Conference (IEPC)*, September 2019.

[48] Anastassios E. Petropoulos and James M. Longuski. Shape-Based Algorithm for the Automated Design of Low-Thrust, Gravity Assist Trajectories. *Journal of Spacecraft and Rockets*, 41(5):787–796, sep 2004.

[49] Lorraine E. Prokop. Core Flight Software Projects on Orion Multi-Purpose Crew Vehicle. December 2018.

[50] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, jan 1999.

[51] Jason A Reiter, David B Spencer, and Richard Linares. Spacecraft detection avoidance maneuver optimization using reinforcement learning. In *29th AAS/AIAA Space Flight Mechanics Meeting, 2019*, pages 3055–3069. Univelt Inc., 2019.

[52] Jason A. Reiter, David B. Spencer, and Richard Linares. Spacecraft Stealth Through Orbit-Perturbing Maneuvers Using Reinforcement Learning. In *AIAA Scitech 2020 Forum*. American Institute of Aeronautics and Astronautics, jan 2020.

[53] 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. IEEE, IEEE, sep 2018.

[54] Sebastian Ruder. An overview of gradient descent optimization algorithms.

[55] Hanspeter Schaub and John L. Junkins. *Analytical Mechanics of Space Systems*. AIAA Education Series. American Institute of Aeronautics and Astronautics, Inc., 3rd edition, 2014.

[56] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *CoRR*, abs/1502.05477, 2015.

[57] 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.

[58] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017.

[59] Andrea Scorsoglio, Roberto Furfaro, Richard Linares, and Brian Gaudet. Image-based Deep Reinforcement Learning for Autonomous Lunar Landing. In *AIAA Scitech 2020 Forum*. American Institute of Aeronautics and Astronautics, jan 2020.

[60] Wolfgang Seefelder. *Lunar Transfer Orbits Utilizing Solar Perturbations and Ballistic Capture*. Herbert Utz Verlag, 2002.

[61] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016.

[62] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, oct 2017.

[63] Jon A. Sims, Paul A. Finlayson, Edward A. Rinderle, Matthew A. Vavrina, and Theresa D. Kowalkowski. Implementation of a Low-Thrust Trajectory Optimization Algorithm for Preliminary Design. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, page 6746. American Institute of Aeronautics and Astronautics, aug 2006.

[64] Jon A. Sims and Steve N. Flanagan. Preliminary Design of Low-Thrust Interplanetary Missions. In *AAS/AIAA Astrodynamics Specialist Conference*, number 99-338, 1999.

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

[66] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press Cambridge, 2nd edition, 2018.

[67] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[68] F. Topputo and C. Zhang. Survey of Direct Transcription for Low-Thrust Space Trajectory Optimization with Applications. In *Abstract and Applied Analysis*, volume 2014, pages 1–15. Hindawi, Hindawi Limited, 2014.

[69] Diogene A. Dei Tos and Francesco Topputo. Trajectory refinement of three-body orbits in the real solar system model. *Advances in Space Research*, 59(8):2117–2132, apr 2017.

[70] Hado van Hasselt. *Reinforcement Learning in Continuous State and Action Spaces*, pages 207–251. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[71] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, feb 2020.

[72] Ryan Whitley and Roland Martinez. Options for Staging Orbits in Cislunar Space. In *2016 IEEE Aerospace Conference*, pages 1–9. IEEE, IEEE, mar 2016.

[73] Brian C. Williams. MIT 16.413 Lecture Slides: Markov Decision Processes: Creating Reactive Policies from Models and Experience. October 2018.

[74] Jacob Williams. Copernicus Trajectory Design and Optimization System. Electronically, August 2018.

[75] Yu-Fen Zhang, Qun-Feng Zhang, and Rui-Hua Yu. Markov property of Markov chains and its test. In *2010 International Conference on Machine Learning and Cybernetics*, volume 4, pages 1864–1867. IEEE, jul 2010.