ABSTRACT

Title of Thesis: COMBINING SUPERVISED PRETRAINING AND REINFORCEMENT LEARNING FOR SCALABLE LOW-THRUST TRAJECTORY DESIGN

Michael Christian Schmidt
Master of Science, 2026
University of Maryland, College Park

Thesis Directed by: Professor John R. Martin
Department of Aerospace Engineering

This thesis considers a hybrid machine learning training framework that combines supervised pretraining with deep reinforcement learning to approximate optimal low-thrust control for heliocentric transfer and rendezvous trajectories. The primary test case is a circular two-body transfer problem in which a continuously thrusting spacecraft is transferred between heliocentric orbits of varying semi-major axis while minimizing propellant consumption. An additional experiment is conducted for an interplanetary rendezvous scenario. Mass-optimal reference trajectories are first generated using an indirect optimal control formulation based on Pontryagin's Maximum Principle and homotopic smoothing to obtain bang-bang thrust profiles. These trajectories are then converted into a Markov decision process dataset by mapping each state and optimal control to a normalized polar state representation and continuous action vector, and encoding them as state–action–reward–transition tuples that populate the experience replay buffer of a Soft Actor-Critic (SAC) agent. Comparative experiments

against a baseline SAC agent trained from scratch show improvements in average episode reward and higher-performing controllers when using pretraining in Monte Carlo validation tests. These results demonstrate that seeding off-policy reinforcement learning with mass-optimal trajectory data is an effective strategy for improving training efficiency in reinforcement learning applied to trajectory design problems.

COMBINING SUPERVISED PRETRAINING AND REINFORCEMENT LEARNING FOR SCALABLE LOW-THRUST TRAJECTORY DESIGN

by

Michael Christian Schmidt

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2026

Advisory Committee:
Professor John R Martin, Chair/Advisor
Professor Robert Michael Sanner
Professor Brent William Barbee

Dedication

I dedicate this work to my grandparents, Michael F. Schmidt (1930–2025) and Sally Schmidt (1935–2023), who never missed a chance to tell their grandkids how proud they were.

Acknowledgments

Many people were essential to the completion of this work, and I am deeply grateful to be surrounded by such wonderful individuals. Over the past few years of graduate study, I've benefited from their kindness, curiosity, and encouragement. I thank them here in no particular order.

First, I would like to thank my Maryland MLDS (Machine Learning for Dynamical Systems) lab mates: Simon Bauer, Kruti Bhingradiya, Kenny Getzandanner, Sean McCurry, Logan Selph, Aarun Srinivas, and Sarah Wielgosz. The early part of my graduate school experience often felt more like a solo journey. Joining this group changed that and I feel like I found my home on campus. At my first meeting, I was intimidated by how sharp everyone was, and I wondered how I would fit in as a part-time master's student among mostly full-time Ph.D. candidates. That uncertainty quickly faded as I realized how welcoming and generous they all were. I am grateful for the friendships I've built in MLDS, and I look forward to many more hikes, Mario Kart, pizza nights, and Spikeball games.

I would also like to express my gratitude to my company, a.i. solutions, for its generous financial support during my graduate studies. I likely would not have pursued a master's degree without the company's tuition assistance program, and I am especially thankful to my managers, Clint Stone and Jeff Dibble, for encouraging me to apply.

Throughout my professional career, I have met many people in the astrodynamics community whose kindness and interest in my work have meant a great deal to me. In particular, I want to thank the Roman Space Telescope flight dynamics team, whom I work with daily and who consistently supported me leading up to and during my defense. Aaron Shepard, Thomas Just, Haijun Shen, Scott Patano, Cassie Webster, and Adam Michaels went out of their way to learn about my research, check in on my progress, and keep my spirits up when I needed it most. I especially value our Friday morning coffee chats, where they listened to me ramble about this work with genuine enthusiasm.

I also owe deep gratitude to Don Dichmann, an astrodynamicist at the NASA Goddard Space Flight Center, whom I met early in my career. Don mentored me when I was a young engineer with little experience and led the Landsat 9 flight dynamics team, the first mission I supported as a flight dynamics engineer. He taught me an enormous amount, from niche technical topics to the broader workings of NASA and the profession. As a University of Maryland alumnus, he also encouraged me when I first expressed interest in graduate school and helped me navigate the application process.

I would like to thank my committee members, Professors Robert Sanner and Brent Barbee, for their patience and understanding throughout the preparation of this manuscript and my thesis defense. Their feedback and guidance during my presentation were invaluable, and I came away with a clearer perspective on both machine learning and astrodynamics—and on the most promising directions for future work.

I have also been fortunate to have exceptional support in my personal life. In my last semester of graduate school, I moved in with two of my closest friends, Andrew and Morgan, during a period of uncertainty for many government contractors and civil servants due to federal budget cuts. They welcomed me on short notice and with open arms. I've loved being their roommate, and I deeply appreciate their generosity and kindness.

I want to thank my girlfriend, Poonam, for her patience, encouragement, and steady support throughout my thesis writing. She has been a steadfast partner when I've struggled and is always ready to talk through an idea, keep me on track, and remind me to persevere. Her selflessness has meant more to me than I can express. She has been a guiding and comforting presence throughout my work—spending countless hours going through my presentations with me and helping me game-plan for deadlines, all while offering a smile or a hug when she could see I needed one.

My parents, Michael and Robin, have been an essential part of this journey as well. They have

never stopped showing their love and support, especially during the many late nights I spent hunched over a computer. Most visits home still involved bringing schoolwork with me, and I am grateful for my dad staying up late to look at trajectory plots with me and for my mom talking me through both the good times and the difficult ones.

Finally, I would like to thank my advisor, Professor John Martin, for his outstanding mentorship throughout this research. I met Dr. Martin when I took the "Decision Making under Uncertainty" course in 2023—my favorite class by far, and one that gave structure to several areas of fascination for me. The principles I learned there form the backbone of much of this thesis. I was surprised to learn it was his first time teaching the course, given how thoughtfully designed and illuminating it was. After the semester, I approached Dr. Martin about combining what I had learned in DMU with trajectory design problems, and he welcomed me into his research group. Since then, he has encouraged me to pursue my research interests, helped me stay focused when I veered off course, and consistently supported my growth with his expertise in machine learning, astrodynamics, software development, and academia.

I am eternally grateful for this experience and for the people around me during my time researching. It would have meant nothing without them.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AU	Astronomical Unit
BEE	Bellman Expectation Equation
CR3BP	Circular Restricted Three-Body Problem
DDP	Differential Dynamic Programming
DDPG	Deep Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
DRO	Distant Retrograde Orbit
ERO	Earth Return Orbiter
GNC	Guidance, Navigation, and Control
HPC	High-Performance Computing
MC	Monte Carlo
MDP	Markov Decision Process
MSR	Mars Sample Return
NLP	Nonlinear Programming
NN	Neural Network
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
PDF	Probability Density Function
PMP	Pontryagin's Maximum Principle
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SB3	Stable-Baselines3
SMA	Semi-Major Axis
TBR	Two-Body Rendezvous Problem
TBT	Two-Body Transfer Problem
TD3	Twin Delayed Deep Deterministic Policy Gradient
TOF	Time of Flight
TPBVP	Two-Point Boundary Value Problem

List of Symbols

Astrodynamics: orbits, geometry, and kinematics

Δv	Total velocity increment
$\mathbf{r}$	Position vector
$\mathbf{v}$	Velocity vector
r	Radial distance (magnitude of $\mathbf{r}$)
r_i	Initial radial distance
r_f	Final radial distance
θ	Argument of latitude
ν	True anomaly
x	Cartesian x position
y	Cartesian y position
v_x	Cartesian x velocity
v_y	Cartesian y velocity
v_r	Radial velocity
v_θ	Tangential velocity
a	Semi-major axis
e	Eccentricity
ω	Argument of periapsis
a_{initial}	Initial-orbit semi-major axis
e_{initial}	Initial-orbit eccentricity
ω_{initial}	Initial-orbit argument of periapsis
ν_{initial}	Initial-orbit true anomaly
a_{final}	Final-orbit semi-major axis
e_{final}	Final-orbit eccentricity
ω_{final}	Final-orbit argument of periapsis
a_t	Target semi-major axis
e_t	Target eccentricity
ω_t	Target argument of periapsis
ν_t	Target true anomaly
r_{tgt}	Target radial distance
$v_{r,t}$	Target radial velocity
$v_{\theta,t}$	Target tangential velocity
L_2	Second collinear Lagrange point

Dynamics and propulsion

m	Spacecraft mass
m_0	Initial spacecraft mass
T	Thrust magnitude
$T_{\max}$	Maximum available thrust

u	Throttle (normalized thrust magnitude)
$\boldsymbol{\alpha}$	Thrust direction unit vector
α_x	x-component of thrust direction
α_y	y-component of thrust direction
α_r	Radial component of thrust direction
α_θ	Tangential component of thrust direction
I_{sp}	Specific impulse
g_0	Standard gravitational acceleration
G	Gravitational constant
μ	Gravitational parameter of the central body
μ_{Sun}	Solar gravitational parameter
$\mathbf{x}$	State vector
$\mathbf{f}$	Dynamics function

Canonical / nondimensional units

l_*	Characteristic length
m_*	Characteristic mass
t_*	Characteristic time
v_*	Characteristic velocity
m'	Nondimensional spacecraft mass
x'	Nondimensional x position
y'	Nondimensional y position
v'_x	Nondimensional x velocity
v'_y	Nondimensional y velocity
r'	Nondimensional radial distance
v'_r	Nondimensional radial velocity
v'_θ	Nondimensional tangential velocity
r'_{tgt}	Nondimensional target radial distance
$v'_{r,t}$	Nondimensional target radial velocity
$v'_{\theta,t}$	Nondimensional target tangential velocity

Time variables

t	Time
t_0	Initial time
t_f	Final time
TOF	Time of flight
t_{tg}	Time-to-go
t'_{tg}	Nondimensional time-to-go

Optimal control and smoothing

λ	Costate vector

λ_r	Costate associated with position
λ_v	Costate associated with velocity
λ_m	Costate associated with mass
H	Hamiltonian
L	Incremental cost (Lagrangian)
J_0	Fuel-optimal performance index
J_1	Energy-optimal performance index
ρ	Switching function
ϵ	Homotopic smoothing parameter
ϵ_i	Initial smoothing parameter
ϵ_f	Final smoothing tolerance
γ_h	Homotopy reduction factor

MDP / RL formulation

s	MDP state
$\mathbf{s}_i$	State vector at step i
$\mathbf{a}_i$	Action vector at step i
r_t	Reward at time step t
d_t	Terminal indicator
e_t	Experience tuple at time step t
$\mathcal{D}$	Replay buffer
π	Policy
$\pi_\theta(a \mid s)$	Stochastic policy parameterized by θ
$U(s)$	State-value function
$Q(s, a)$	Action-value function
γ	Discount factor
G_t	Return (discounted sum of rewards)

SAC-specific

α	SAC temperature parameter
$\mathcal{H}(\pi)$	Policy entropy
$\mathcal{H}_{\text{tgt}}$	Target entropy
y_t	Soft Bellman target
J_Q	Critic loss
J_π	Actor objective
$J(\alpha)$	Temperature (entropy) objective
τ	Target-network smoothing coefficient

Training and evaluation bookkeeping

w_p	Position-reward weight
w_v	Velocity-reward weight

w_u	Throttle-penalty weight
$\Delta r'$	Nondimensional radial distance residual
$\Delta v'$	Nondimensional velocity residual
$\mathbf{\Delta r}'$	Nondimensional position residual vector
$\mathbf{\Delta v}'$	Nondimensional velocity residual vector
N_{ep}	Number of episodes per evaluation rollout
N_{MC}	Number of Monte Carlo rollouts
N_{pt}	Number of supervised pretraining gradient updates
N_{rl}	Number of reinforcement learning training updates
N_{steps}	Training steps/iterations
$\bar{R}$	Mean episode reward
σ	Standard deviation of episode reward
η	Learning rate
N_{batch}	Training batch size
N_{buf}	Replay buffer capacity

Chapter 1: Introduction

Machine learning applied to trajectory design problems is a burgeoning field of research and has shown promise in the literature as an alternative to optimal control. The approaches for applying these methods are numerous, and the research community has yet to establish a unified training pipeline that produces robust and efficient trajectory design solutions. Some approaches leverage supervised learning—training networks to emulate low-thrust optimal controllers—while others pose the trajectory design problem as a Markov decision process and apply reinforcement learning (RL) methods. The former can yield faster training times when prior data are available, whereas the latter typically requires longer training but often produces controllers that generalize more effectively to perturbations in the initial conditions. A gap in the literature remains regarding how hybrid training approaches can balance the strengths of both paradigms while avoiding the drawbacks of relying exclusively on either one. Namely, this work posits that it is possible—and beneficial—to seed reinforcement learning trajectory design agents with supervised pretraining to significantly reduce training time and improve generalizability.

This thesis explores a hybrid learning approach that combines supervised learning and reinforcement learning to approximate solutions to low-thrust optimal trajectory design problems. Prior work has demonstrated that machine learning can be used to emulate optimal-control mappings, augment established numerical optimization pipelines, and reduce the computational cost of generating low-thrust trajectories. In practice, however, many demonstrated applications are tailored to specific mission scenarios or precisely defined problem classes—for example, neural-network-based trajectory correction [5], compensation for missed-thrust events [6], or low-thrust transfers between libration orbits in the Earth–Moon circular restricted three-body problem (CR3BP) [7].

Extending these methods toward more diverse, target-generalized mission design or control agents is challenging because performance often scales with the availability of large, high-quality training datasets (for supervised learning) or extensive simulation experience (for reinforcement learning). As problem complexity increases—through longer time horizons, tighter terminal constraints, or broader distributions of initial and target conditions—the required data volume and compute budget can quickly become prohibitive. The central premise investigated in this work is that a hybrid pipeline can mitigate these costs by using supervised pretraining to initialize the policy and value-function approximators, followed by reinforcement learning to preserve exploration and refine performance under the full task objective. This combination is intended to retain the strengths of both paradigms while reducing the practical burden associated with using either one in isolation.

Accordingly, this study evaluates the extent to which supervised pretraining on fuel-optimal reference trajectories can reduce the sample complexity of Soft Actor-Critic RL, while balancing terminal accuracy and propellant efficiency for the Two-Body Transfer and Two-Body Rendezvous problems.

1.1 Motivation

There are distinct advantages to using hybrid machine learning methods to approximate low-thrust control in certain settings. Low-thrust propulsion is becoming increasingly common in modern spacecraft designs because it enables much higher effective Δv with modest propellant costs, supporting new mission architectures and reducing mission costs. Low-thrust spacecraft operate by thrusting continuously (or quasi-continuously) over long arcs, and designing efficient trajectories with traditional optimal control methods can be computationally expensive. Controllers trained via machine learning—especially neural-network-based controllers—offer promise in this area. Once

trained, neural-network controllers can be evaluated in microseconds, making them negligible-cost surrogates for onboard control relative to iterative re-optimization [5]. The need for autonomous guidance, navigation, and control in both onboard and ground-based flight dynamics systems is driven by the increasing rate at which new spacecraft are launched and operated. Small satellites and large constellations, in particular, could benefit from more autonomous control systems that can independently maintain or reconfigure their trajectories and potentially avoid collisions. This thesis is motivated by the need for methods that retain the physics-consistent accuracy and structure of optimal control solutions while also enabling fast, closed-loop guidance suitable for real-time correction and scalable replanning.

Beyond the computational expense of optimal control, low-thrust mission operations introduce uncertainty and autonomy constraints that motivate machine learning approaches to control. In practice, trajectories must be corrected under model mismatch and dispersions (e.g., navigation error, thrust performance uncertainty and misalignment, etc.), which pushes the control architecture toward frequent updates rather than a single pre-generated optimal solution. At the same time, physical and logistical limitations such as deep-space communication delays, limited ground-station visibility opportunities, and staffing or resource constraints motivate fast onboard control. Machine-learning controllers can map the current state and target conditions to feasible control actions while mitigating these constraints.

Machine learning can provide fast, closed-loop control that supports increased autonomy in space and scalable replanning on the ground. Optimal control, by contrast, offers high-fidelity, physically consistent, and interpretable solutions for trajectory design. This work argues that these approaches are complementary rather than competing: each is valuable in its intended role, and neither is universally superior. The long-term trajectory design and spacecraft control landscape will likely

benefit from hybrid pipelines that combine optimal control structure with learned, real-time decision policies.

1.2 Problem Statement and Scope

This work explores the application of hybrid supervised and reinforcement learning techniques to two classes of problems: the Two-Body Transfer problem (TBT) and the Two-Body Rendezvous problem (TBR). Each of these astrodynamics problems involves a spacecraft equipped with a low-thrust propulsion system maneuvering in interplanetary space. The TBT involves maneuvering a spacecraft from an initial Keplerian orbit to a target Keplerian orbit after a fixed time of flight. In the TBT, the final true anomaly at which the target orbit is achieved is unconstrained. The TBR involves maneuvering to a target orbital state that includes a specified true anomaly after a fixed time of flight, increasing the number of terminal constraints by one relative to the TBT.

While the TBT and TBR problems have been extensively studied in the literature and their trajectory optimization methods are well understood, they serve as useful benchmarking environments for testing the benefits of hybrid machine learning approaches. The focus of this thesis is to develop a pretraining pipeline and quantify its effects on these established problems, while minimizing the confounding effects of introducing too many new variables at once. By using these well-studied problems, this work can leverage available mass-optimal solutions to compare the performance of machine learning controllers against.

For these problems, the only gravitational body considered in the dynamics is the Sun, and the transfer and rendezvous trajectories are constrained to the inner solar system. Multi-body effects, orbital perturbations, and other dynamics that affect the trajectory are not considered in this work. The

spacecraft state used in both the supervised and reinforcement learning processes is assumed to be known perfectly. The motion of the spacecraft is also simplified to two spatial dimensions (i.e., x and y).

This work makes use of the Soft Actor-Critic (SAC) reinforcement learning algorithm. SAC was selected due to specific features that simplify experimentation with pretraining. Later sections explain why these features facilitate combining supervised pretraining with RL; however, this work does not claim that SAC, or actor-critic methods more broadly, are optimal for this approach.

For this investigation, the goal is to design fuel-optimal trajectories that achieve a desired target state. This analysis does not include an explicit optimization of other objectives, such as minimizing time of flight. This work is intended as a study to evaluate the potential benefits of using supervised pretraining with RL to produce controllers that approximate optimal control quickly; it is not intended as a replacement for comprehensive mission design. The central question is whether supervised pretraining using mass-optimal trajectory datasets reduces the sample complexity of SAC while preserving (or improving) terminal accuracy and propellant efficiency.

1.3 Key Challenges

Several challenges arise when evaluating the effects of supervised pretraining combined with reinforcement learning for low-thrust trajectory design. A primary difficulty is the long-horizon, continuous-control nature of low-thrust optimization. To apply RL, the trajectory problem must be posed as a Markov Decision Process (MDP), which immediately raises a credit-assignment challenge: the agent must satisfy stringent terminal conditions while also minimizing an objective accumulated over the trajectory (e.g., propellant usage). In practice, terminal-state performance is the most mean-

ingful indicator of success, but relying too heavily on terminal evaluation can lead to sparse reward signals and unstable learning. To mitigate this, the pretraining and reinforcement learning phases in this work use identical, fixed propagation step sizes. While step-size invariance is desirable (i.e., controllers that generalize across a range of integration steps), accommodating variable step sizes would introduce an additional input dimension and substantially increase the data requirements. Accordingly, this work assumes a fixed step size in both supervised learning and RL; the reference ephemerides are interpolated to align with the chosen step size used to form state–action transitions.

Closely related to sparse terminal rewards is the challenge of reward shaping. The reward signal must reflect the desired behavior——accurate terminal targeting with efficient propellant use——without enabling "reward hacking," in which the agent exploits the reward structure in ways that do not correspond to true mission objectives. In this work, the reward functions for the TBT and TBR problems are designed to provide dense, continuous feedback throughout the time of flight to reduce the learning difficulties associated with purely terminal rewards. However, this design introduces weighting parameters that require tuning. The selected values were obtained through iterative testing, but they are unlikely to be globally optimal. A more systematic hyperparameter sweep over reward weights would likely improve training stability and could also improve alignment between RL behavior and the optimal-control demonstrations used for pretraining. Balancing dense, informative shaping against faithful alignment with the underlying optimal-control objective remains a central challenge and an active area of experimentation.

A further challenge in the data generation process, supervised pretraining, and RL fine-tuning is maintaining consistency of dynamics and constraints across phases. The dynamics used to generate reference trajectories must match the state-transition model used by the learning environment; even subtle mismatches can degrade performance and confound comparisons. The reference dataset must

also be high fidelity and sufficiently diverse, covering the relevant state space without introducing unintended biases in orbital regimes, time-of-flight distributions, or control modalities. Ensuring both fidelity and coverage is particularly important because the pretraining dataset can strongly influence the initial policy and value estimates.

Finally, effectively combining supervised learning and reinforcement learning requires preserving exploration when a locally attractive optimum is present. Preserving exploration is especially important in this work because the primary RL algorithm is Soft Actor-Critic (SAC), which learns a stochastic policy using entropy-regularized optimization. During pretraining——when updates are driven by demonstration data rather than on-policy interaction——there is a risk that the learned policy becomes overly confident and low-entropy (near-deterministic), particularly if the dataset is limited in coverage. Such entropy collapse can reduce state–action visitation during subsequent environment interaction, slowing adaptation to off-demonstration states and harming final performance. Because SAC relies on policy entropy to sustain exploration, supervised pretraining must be structured to maintain sufficient stochasticity so that the agent can continue to explore and improve during RL fine-tuning.

1.4 Proposed Approach

The high-level approach used in this work consists of five major steps for both the TBT and TBR problems. First, fuel-optimal reference trajectories are generated to serve as training data for supervised learning. Next, the TBT and TBR problems are cast as Markov decision processes for compatibility with reinforcement learning (RL) algorithms. Once the training data are generated and the MDP formulations are in place, supervised learning is performed to initialize the policy and value networks of an RL agent. In this analysis, the Soft Actor-Critic (SAC) algorithm is used for the

RL portion of training, and the supervised pretraining stage is designed to align with SAC's network architecture. Specifically, pretraining is accomplished by performing gradient-based updates to the policy and value networks using only the reference trajectories. The pretrained agent is then passed into a standard SAC training process, using the pretrained policy and value networks as an initial approximation of fuel-optimal control. As a result, instead of executing purely random actions at the beginning of SAC training, the policy produces control inputs consistent with the reference training data while still exploring new trajectories and previously unseen states. These new trajectories are stored in the experience replay buffer as they are encountered and are used to update the agent policy as additional states are observed. SAC further encourages exploration through an entropy-regularization mechanism while refining the pretrained policy and value estimates.

After pretraining and RL, a Monte Carlo evaluation process is used to characterize agent performance. Random initial conditions are sampled from the same orbital distributions used for the training data and reinforcement learning, and the agent is evaluated on performance over these previously unseen trajectories. Statistics are gathered for the total reward accrued by the agent, as well as terminal position and velocity residuals and delivered spacecraft mass at the target. Figure 1.1 illustrates this analysis workflow.

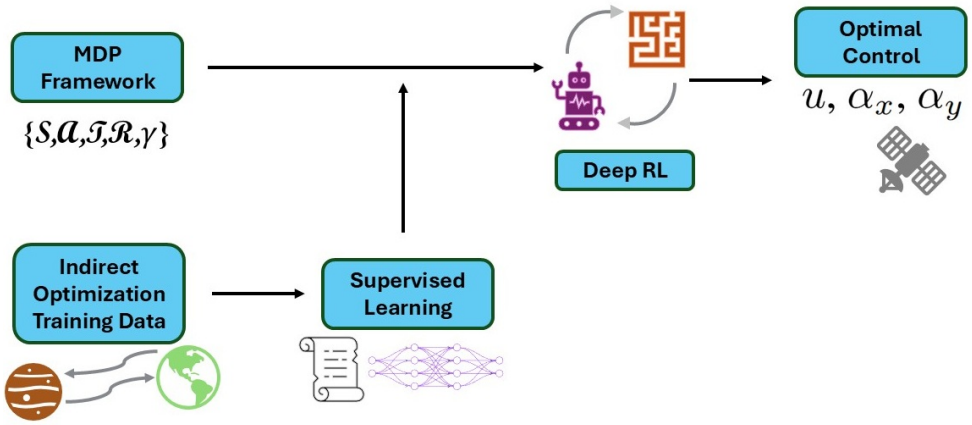


Figure 1.1: Hybrid Supervised and Reinforcement Learning Pipeline

1.5 Contributions

This thesis presents the following technical contributions:

1. Introduced a supervised pretraining stack to RL applied to low-thrust trajectory design.
2. Established two baseline low-thrust Markov Decision Processes readily incorporated into continuous RL algorithms to quantify the impact of supervised pretraining.
3. Presented empirical results from preliminary sensitivity studies on the MDP environments measuring the impacts of supervised pretraining.

1.6 Thesis Organization

This thesis is organized as follows. Chapter 2 explores the state of the literature in low-thrust trajectory design and modern applications of optimization methods. The literature review also examines applications of supervised machine learning and reinforcement learning in the low-thrust trajectory design domain.

Chapter 3 covers the technical background necessary to discuss the hybrid machine learning approach to low-thrust problems used in this work. It provides an overview of optimal control methods for low-thrust trajectory design, with emphasis on the indirect methods used to generate training data for later experiments. An introduction to reinforcement learning is then provided, followed by a discussion of deep reinforcement learning and supervised learning with neural networks. The chapter concludes with a detailed description of the Soft Actor-Critic algorithm, the primary RL method applied to trajectory design problems in this thesis.

Chapter 4 presents the methodology behind the hybrid learning approach. It describes the generation of fuel-optimal trajectories used to construct the primary datasets in this thesis. Markov decision process (MDP) formulations are provided for the TBT and TBR problems, along with an explanation of how the trajectory data are converted into replay-buffer experiences for supervised pretraining. The chapter then summarizes how these components are integrated into a supervised pretraining pipeline and concludes with the Monte Carlo evaluation approach used during and after training.

Chapters 5 and 6 present results from the two primary experiments in this work. Chapter 5 describes a proof-of-concept experiment that demonstrates the utility of pretraining. Chapter 6 reports results for a more challenging transfer problem and examines the combined effects of pretraining data volume and the number of pretraining iterations on agent performance.

Chapter 7 discusses the experimental results and identifies promising directions for future work. Chapter 8 concludes the thesis by summarizing the empirical findings and providing closing remarks on combining supervised learning and reinforcement learning for trajectory design problems.

Chapter 2: Literature Review

2.1 Low-Thrust Trajectory Optimization

Low-thrust trajectory design is commonly formulated as an optimal control problem (OCP) involving a spacecraft with an allowable set of control inputs that must be guided from an initial state to a target state while optimizing a performance metric. Common optimization metrics include minimizing propellant consumption, minimizing time of flight, or some weighted combination of the two. Classical approaches to solving this problem treat it as a constrained optimization over the spacecraft's control time history, subject to the dynamics and boundary conditions of the problem under consideration. Pontryagin's Maximum Principle (PMP) provides the foundational necessary conditions for optimality through the use of costates and a Hamiltonian. Costates in this context are analogous to Lagrange multipliers [8], and the Hamiltonian is constructed from the problem states, costates, and performance metric; minimizing it yields the optimal control along a trajectory [9].

Modern references focus on the practical numerical strategies involved in applying PMP-based methods to low-thrust trajectory design problems, including sensitivity to initial costate guesses, efficient solution of the two-point boundary value problem (TPBVP) induced by these methods, handling of discontinuous thrusting behavior, and convergence limitations [10, 11].

Two traditional approaches discussed in the literature for solving low-thrust trajectory problems are *indirect* and *direct* methods. The PMP-based approach described above is categorized as an indirect method because it solves for the costates, which are then used to compute the optimal control along a trajectory [9]. Direct methods instead transcribe the OCP into a finite-dimensional nonlinear programming (NLP) problem by parameterizing the state and control and enforcing the dynamics

through constraints. Direct methods do not require the explicit computation of costates and often offer increased robustness and greater flexibility with respect to inequality constraints; however, this typically comes at the cost of higher decision-variable dimensionality and reliance on comparatively heavy NLP solvers [10, 11]. A representative early direct strategy for preliminary low-thrust mission design is the Sims–Flanagan method, which discretizes the trajectory into segments and enforces boundary-matching constraints, enabling efficient exploration of mission trades during early-phase design [12]. Modern approaches build on this strategy and use collocation schemes that improve accuracy and adaptability through careful selection of nodes. For example, Holden *et al.* [13] use adaptive quadrature collocation for interplanetary transfers to refine the transcription (the discretized representation of the problem) where needed.

Direct methods are generally more robust to initial guesses, whereas indirect methods tend to converge to tighter tolerances [14]. The robustness of direct methods enables solution of trajectory design problems with more complicated boundary conditions, but this typically comes at increased computational cost. The optimization process used in this work is an indirect strategy that solves for costates leading to optimal control. One reason for selecting an indirect method is the relatively simple, well-known analytical expressions for the costates in the TBT and TBR problems investigated here. Sidhoum and Oguri [3] provide compact solutions that can be readily incorporated into ordinary differential equation (ODE) solvers used to generate optimal trajectories. Indirect methods also provide a guarantee of optimality with respect to the problem constraints when the PMP necessary conditions are satisfied [9], whereas direct methods do not provide an explicit optimality guarantee without additional analysis [15]. Indirect methods can also converge to comparatively tighter tolerances than direct methods because they involve fewer decision variables. In addition, indirect methods enforce the continuous dynamics throughout the trajectory, avoiding transcription-induced artifacts

and yielding higher-fidelity reference trajectories for the training datasets.

Between purely indirect single shooting and fully collocated direct transcription lies a spectrum of hybrid approaches. Multiple shooting partitions the time horizon into subarcs, integrates the dynamics on each arc, and enforces continuity via matching constraints. This can improve conditioning relative to single shooting while preserving the favorable structure of integrating the dynamics directly [10,11]. Meng *et al.* demonstrate low-thrust minimum-fuel optimization using multiple shooting augmented by analytical derivatives, highlighting both the practical viability of shooting-style decompositions and the importance of accurate sensitivity information for convergence and computational efficiency [16]. Such methods are especially relevant for problems where dynamics integration is relatively inexpensive (e.g., two-body models) and where solver performance can benefit from structured Jacobians and derivative information.

2.2 Modern Approaches to Indirect Optimization in Low-Thrust Spacecraft Control

Indirect methods remain an active area of research for low-thrust trajectory design because many minimum-fuel problems exhibit a bang–bang optimal throttle structure. Bang–bang control is a regime in which the spacecraft either thrusts at its maximum capacity or is off entirely, with no smooth transitions in between [17]. This structure can make TPBVP shooting numerically fragile; consequently, modern work often focuses on improving robustness and convergence without abandoning the Pontryagin-based structure that makes indirect methods attractive.

A common practice is to introduce smoothing or continuation adjustments to regularize discontinuous control behavior and expand the convergence region of TPBVP solvers. Bertrand and Epenoy present a set of smoothing techniques for bang–bang optimal control problems and provide numerical

results and interpretation that motivate continuation-style approaches for reliable convergence [18]. Building on this idea in a fuel-minimization setting, Taheri *et al.* put forth an enhanced smoothing technique tailored to indirect optimization of minimum-fuel low-thrust trajectories, demonstrating how smoothing-method selection can affect convergence behavior and solution quality [19].

Beyond bang–bang discontinuities, another challenge for indirect optimization methods is sensitivity to the initial costate guess. Parrish *et al.* provide an applied example of indirect low-thrust optimization in a challenging dynamical environment (Earth–Moon DRO-to-L_2 halo transfers), illustrating both the practicality of indirect methods and the sensitivity that motivates improved costate initialization and continuation practices [20]. To address these initialization difficulties directly, Hecht and Botta introduce a particle swarm optimization (PSO) approach for costate initialization in low-thrust minimum-fuel problems, using a global-search heuristic to improve convergence of the subsequent indirect solution [21].

Recent efforts also explore hybrid formulations that preserve PMP structure while incorporating ideas from dynamic programming or second-order trajectory improvement methods, particularly to better handle path constraints and improve numerical stability. Sidhoum and Oguri present a Pontryagin–Bellman differential dynamic programming framework for low-thrust trajectory optimization with path constraints, representing a modern direction that blends indirect necessary conditions with dynamic programming updates [22].

In this thesis, the indirect solution approach is used primarily as a reliable generator of high-quality, mass-optimal reference trajectories for supervised pretraining and benchmarking. Because of the simple two-body dynamics used in the equations of motion, a relatively compact analytic structure exists for the optimal control; therefore, the emphasis is placed on smoothing and continuation for robust convergence rather than more elaborate enhancements such as global-search costate ini-

tialization or hybrid DDP methods. These modern techniques provide a clear path for extending the data-generation pipeline to more complex dynamics and constraints in future work [18–22].

2.3 Supervised Learning Applied to Spacecraft Guidance and Control

Supervised learning provides a practical way to approximate expensive mappings that arise in trajectory design and guidance by training neural networks on labeled data generated from physics-based simulation or numerical optimization. Modern deep networks serve as complex function approximators that can be trained with gradient-based methods, enabling rapid inference after training [23]. This section will focus on supervised learning specifically as it has been applied to spacecraft control and low-thrust trajectory problems, and how supervised learning can be used to as an enhancement or a substitute for indirect and direct trajectory optimization methods.

From a controls perspective, neural networks can be used to approximate nonlinear mappings needed for optimal low-thrust control provided representative training data are available and care is taken with generalization and closed-loop behavior [24]. For this work, this perspective motivates the use of supervised learning to approximate the optimal low-thrust control structure learned from reference solutions generated with indirect methods, and will provide informed initialization for downstream reinforcement learning. General deep-learning concepts follow standard treatments when applied to trajectory design problems as explained by Goodfellow, Bengio, and Courville [23].

A useful view is to treat deep learning based trajectory optimization methods as a number of subcategories as described in Li, Jia, and Zhang: (i) surrogate modeling of costs/metrics to accelerate search and screening, (ii) direct approximation of guidance/control laws for real-time use, and (iii) learned components that assist classical optimization, such as warm-starting, continuation, and correc-

tion [25]. This taxonomy provides a structure for the remainder of this subsection and clarifies where the contributions of the present work sit relative to prior studies.

Category I: surrogate models for rapid cost/feasibility evaluation. A common supervised-learning use case is to replace repeated numerical optimization calls with fast predictors of mission-relevant quantities like transfer cost, time, or feasibility proxies. For example, deep networks have been trained to approximate optimal low-thrust cost quantities in multi-target settings, enabling rapid evaluation that supports higher-level mission design trade studies [26]. This line of work emphasizes increased scalability and output. Once a network is trained to estimate these mission-relevant quantities, a surrogate network can be queried orders-of-magnitude faster than solving an optimal control problem for each candidate solution.

Related efforts extend supervised learning beyond single transfers toward mission-design contexts involving multiple rendezvous targets and continuous-thrust constraints. Viavattene and Ceriotti, for instance, apply neural networks to multiple-NEA rendezvous mission design with continuous thrust, demonstrating how learned models can support complex design spaces that would otherwise require extensive optimization runs [27]. Considered alongside the mission feasibility proxy estimation work, these studies motivate supervised learning as a way to compress expensive trajectory design computations into reusable models.

Category II: end-to-end guidance and real-time low-thrust control approximation. A second major theme is using supervised learning to approximate the optimal control law itself: mapping a state (and implicitly, target and time-to-go) to the control vector suitable for low-thrust transfer. Real-time guidance is a particularly strong motivation, since onboard or interactive applications demand fast

inference once training is complete. Izzo and Öztürk demonstrate real-time guidance for low-thrust transfers using deep neural networks, illustrating how learned policies can approximate solutions that would otherwise require repeated numerical optimization [28], as is required in direct and indirect optimization methods of the previous subsection. This category is closest conceptually to treating the neural network as an explicit controller instead of a tool that facilitates offline design.

Category III: correction and off-nominal recovery using supervised learning. Supervised learning has also been used to correct trajectories or recover performance when the spacecraft experiences deviations from a nominal plan. Rubinsztejn *et al.* study neural-network optimal control in an astrodynamics setting with application to the missed-thrust problem, demonstrating how networks can be trained to produce corrective actions (or corrected control) when thrust execution differs from a baseline reference [6]. This is particularly relevant to practical operations, where low-thrust missions may face execution errors, modeling mismatch, or uncertainty that motivates rapid correction and looking past the idealized scenarios of early-phase mission design.

As mentioned in Subsection 2.1, a persistent practical challenge in low-thrust trajectory optimization is generating sufficiently good initial guesses for indirect or shooting-based methods. Parrish and Scheeres highlight this difficulty and propose approaches for low-thrust optimization with no initial guess, motivating the role of learned components as a way to reduce dependence on user-provided initialization [29]. In follow-on work, neural networks are used explicitly for optimal low-thrust trajectory correction, reinforcing the idea that supervised models can be integrated with traditional solvers to improve robustness and reduce iteration counts [5].

Across the three major categories of supervised learning identified, a consistent pattern emerges: supervised learning is most effective when (i) high-quality labeled data are available (often from opti-

mal control solvers), and (ii) the learned model is embedded in a workflow that exploits its strengths while managing its limitations [23,25]. In the context of this thesis, these studies motivate using mass-optimal low-thrust reference solutions as training data for initial supervised learning that will take place prior to reinforcement learning for low-thrust trajectory design. To distinguish this supervised training process as the first segment of a two-part deep learning process (to be followed by reinforcement learning), it will be referred to as pretraining in this specific application.

2.4 Reinforcement Learning in Spacecraft Guidance and Control

Kochenderfer provides a deep and clear foundation for general reinforcement learning in a variety of applications [1], and covers many aspects of decision making agents and planning used in this work. The foundational aspects of reinforcement learning used in this work are drawn from this reference. These principles are leveraged by many works in low-thrust trajectory design in a variety of settings.

A comprehensive survey of the use of deep RL in spacecraft control applications as of 2022 is provided by Tipaldi, Iervolino, and Massenio [30]. A wide variety of approaches covered in literature are summarized in this work in many aspects of spacecraft control. From this, a few particular works were analyzed in detail in support of this analysis. LaFarge *et al.* [7] demonstrate that modern policy-optimization methods can produce closed-loop low-thrust guidance policies in multi-body environments that are difficult to address with classical two-body analytic structure alone. Their results highlight one of the core motivations for RL in spacecraft guidance: rather than solving a new optimization problem from scratch for each initial condition, a trained policy can provide fast feedback control that generalizes across a distribution of initial states and perturbations. Holt *et al.* [31] inves-

tigate actor–critic-style learning for low-thrust trajectory design and emphasize the practical value of learning feedback-driven control laws with state-dependent parameters. Together, these works illustrate a recurring theme in RL for spacecraft control: RL is often most useful when the goal is not to replace optimal control solvers all together, but to provide deployable closed-loop guidance that can be evaluated quickly and repeatedly under uncertainty, model mismatch, and there is a large-scale demand of trajectories.

Within the broader RL landscape summarized by Tipaldi *et al.* [30], actor–critic methods are especially prevalent for spacecraft problems because they scale well to continuous states and actions and can exploit function approximation without requiring explicit value tables. In this thesis, the Soft Actor–Critic (SAC) algorithm is selected as the primary RL approach because it is an off-policy actor–critic method designed for continuous control with an explicit maximum-entropy objective [32]. Haarnoja *et al.* [32] is a central work used in understanding the features of SAC RL and how they can be leveraged for supervised pretraining.

A key bottleneck in continuous-control RL is that naïve exploration can be prohibitively expensive when dynamics are long-horizon and constraints are tight. Nair *et al.* [33] show that incorporating demonstration data can materially reduce the exploration burden in actor–critic training by biasing early learning toward competent behavior while still allowing improvement through reinforcement learning. They demonstrate this for robotic block stacking problems, the central ideas explored are analogous to what is investigated here in a spacecraft guidance application.

From a broader perspective, Levine *et al.* [34] formalize the setting in which policies and value functions are trained from a fixed dataset (offline RL), highlighting both the promise of leveraging pre-generated trajectories and the central risks associated with distribution shift and extrapolation error when the learned policy visits states that are weakly represented in the dataset. In response to these

challenges, methods that explicitly blend imitation and off-policy RL objectives have emerged; for example, Fujimoto and Gu [35] propose a simple approach that augments value-based actor–critic updates with a behavior-cloning term to keep the learned policy near the data distribution while still enabling improvement through reinforcement learning.

Taken together, these results motivate the hybrid pipeline used in this work: fuel-optimal reference trajectories can be interpreted as expert demonstrations that initialize the policy and value estimates, after which off-policy RL can refine performance through continued interaction and exploration. In particular, SAC's replay-buffer training structure makes it straightforward to incorporate pre-generated optimal-control trajectories as initial experience, and its entropy-regularized objective provides a principled mechanism to preserve exploration while the agent transitions from imitation-dominated learning to online reinforcement learning [32]. This framing positions supervised pretraining not as a replacement for RL, but as an initialization strategy that improves sample efficiency and stabilizes early learning, consistent with the motivation emphasized in demonstration-augmented and offline RL studies [33–35].

Chapter 3: Technical Background

3.1 Optimal Control

3.1.1 Low-Thrust Problem Formation

There are two classes of problems considered in this analysis. The first is the "Two-Body Transfer" problem, which refers to maneuvering a spacecraft from one circular orbit to another circular orbit with a different semi-major axis. This problem is confined to two spatial dimensions, x and y. The second is the "Two-Body Rendezvous" problem, which involves maneuvering a spacecraft from an initial circular orbit to match the state of a specified target after a fixed time of flight. For both problems, the central body is assumed to be the Sun. The gravitational effects of other planets and celestial bodies in the solar system are neglected; the Sun is the only gravitating body acting on the spacecraft.

While these two problems are distinct in their respective terminal conditions, the dynamics governing the spacecraft motion are the same in both the Two-Body Transfer (TBT) and Two-Body Rendezvous (TBR) cases. The Cartesian equations of motion are given in Eq. (3.1).

$$\dot{\vec{x}} = \vec{f}(\vec{x}, u, \vec{\alpha}) = \begin{bmatrix} \dot{\vec{r}} \\ \dot{\vec{v}} \\ \dot{m} \end{bmatrix} = \begin{bmatrix} \vec{v} \\ -\frac{\mu}{r^3}\vec{r} + \frac{T_{\max} u}{m}\vec{\alpha} \\ -\frac{T_{\max} u}{I_{sp} g_0} \end{bmatrix} \tag{3.1}$$

This system includes two control inputs: the thrust throttle and the thrust direction. The thrust throttle, denoted by u, ranges from 0 to 1 and dictates what fraction of the maximum available thrust the spacecraft applies while maneuvering. The thrust direction, $\vec{\alpha}$, is the unit vector along which thrust

is applied. Because the motion is confined to two dimensions in this simulation, the state vector has five components and the control vector has three components: u, α_x, and α_y.

3.1.2 Canonical Coordinate Encoding

Both the optimization processes used to generate the training data and the reinforcement learning algorithms discussed later make use of a canonical coordinate encoding. This amounts to transforming the state vector into nondimensional units and is intended to prevent computational issues that can arise when integrating equations of motion with state components of vastly different magnitudes. These nondimensional encodings are also useful when querying neural networks that accept state vectors as inputs. In practice, neural network training is often improved when input features are of comparable scale and ideally lie between -1 and 1.

The transformation to canonical units follows the standard set of expressions in [36]. First, the characteristic length of the coordinate system is defined as the semi-major axis of Earth's orbit: $l_* = a_{\text{Earth}} = 1 \text{ AU}$. The characteristic mass of the system is defined as the mass of the central body, $m_* = m_{\text{Sun}}$. The final piece of the transformation is the characteristic time. Consider a circular reference orbit with semi-major axis equal to l_*. The orbital velocity and subsequent characteristic time are:

$$\begin{aligned} v_* &= \sqrt{\frac{\mu_*}{l_*}} = \frac{l_*}{t_*} \\ t_* &= l_* \sqrt{\frac{l_*}{\mu_*}} = \sqrt{\frac{l_*^3}{\mu_*}} \end{aligned} \tag{3.2}$$

This leads to the following quantities used in this work for nondimensionalization during nu-

merical integration and neural network inference:

$$
\begin{aligned}
l_* &= a_{\text{Earth}} \approx 1.496 \times 10^{11}\ \text{m}, \\
m_* &= m_{\text{Sun}} \approx 1.988 \times 10^{30}\ \text{kg}, \\
t_* &= \sqrt{\frac{l_*^3}{G m_*}} \approx 5{,}023{,}281\ \text{s}, \\
G &= 6.674 \times 10^{-11}\ \frac{\text{m}^3}{\text{kg}\,\text{s}^2}.
\end{aligned}
\tag{3.3}
$$

These quantities are then used to non-dimensionalize the state vector components:

$$
\begin{aligned}
x' &= \frac{x}{l_*}, \\
y' &= \frac{y}{l_*}, \\
v_x' &= \frac{v_x t_*}{l_*}, \\
v_y' &= \frac{v_y t_*}{l_*}.
\end{aligned}
\tag{3.4}
$$

The mass of the Sun is used to define a sensible characteristic time; however, it is not a suitable quantity for nondimensionalizing spacecraft mass due to the extreme difference in magnitudes. Therefore, spacecraft mass is normalized by the initial spacecraft mass (prior to propellant expenditure):

$$
m' = \frac{m}{m_0} \tag{3.5}
$$

A quick sanity check of these quantities is warranted. Using these non-dimensional units is akin to setting the gravitational constant G equal to one [36] and operating in a transformed dynamics problem. Substituting each of the non-dimensional quantities into the units of the gravitational constant should lead to its known empirical value of $6.674 \times 10^{-11} \frac{m^3}{kg \times s^2}$. A quick assessment of these units confirms this:

$$G_* = \frac{l_*^3}{m_* t_*^2} = \frac{(1.496 \times 10^{11} m)^3}{(1.988 \times 10^{30} kg) \times (5,023,281 s)^2} = 6.674 \times 10^{-11} \frac{m^3}{kg \times s^2} \tag{3.6}$$

3.1.3 Optimal Control Problem Definition

This analysis is concerned with navigating from an initial state to a target while minimizing fuel mass. A cost function, also referred to as a performance metric, will be defined for the control problems that will give a quantitative measure of how optimal a given trajectory is. Consider a trajectory propagated using the equations of motion given in Equation (3.1) from an initial time t_0 to a final time t_f, subject to control by $u(t)$ and $\boldsymbol{\alpha}(t)$ during this interval. A performance metric that will yield a mass-optimal trajectory (meaning that the final mass of the spacecraft at t_f should be maximized) is desired and can be shown to be:

$$J_0 = \frac{T_{\max}}{I_{sp} g_0} \int_{t_0}^{t_f} u(t)\, dt \tag{3.7}$$

This integral comes from the fact that the mass rate should be minimized throughout the trajectory as much as possible during the flight time. Since the mass derivative from Equation (3.1) is just a function of the maximum spacecraft thrust, thruster specific impulse, gravitational acceleration at the Earth's surface, and the throttle input; the constants can be pulled out of the integral and the perfor-

mance index becomes just an integral of u over the time span of interest. The goal of the optimization process will be to minimize J_0 to yield an optimal trajectory.

The control variables consist of the throttle control u and the burn direction vector $\boldsymbol{\alpha}$. The control throttle is a scalar constrained between 0 and 1 and dictates the instantaneous fraction of the maximum available thrust the spacecraft can maneuver with. The burn direction is the unit vector for which the thrust is applied. These constraints are provided below in condensed form:

$$\begin{aligned} 0 \leq u \leq 1 \\ \|\alpha\| = 1 \end{aligned} \tag{3.8}$$

While the performance index J_0 is ultimately what should be minimized in the optimal control problems discussed here, another performance index; J_1, will be defined because it is a useful stepping stone in computing trajectories due to numerical issues involved with handling the mass-optimal performance index J_0. Equation (3.9) is the "energy-optimal" cost function, where the integrand is the square of the throttle input u.

$$J_1 = \frac{T_{\max}}{I_{sp} g_0} \int_{t_0}^{t_f} u^2(t)\, dt \tag{3.9}$$

It should be noted here that for the control problems being considered in this analysis, the time of flight ($TOF = t_f - t_0$) will be fixed during the propagation and will be considered an input parameter. Optimizing the time of flight, among other design variables, are of interest in this process but are not covered in this work. Refer to Kim [11] for further reading on additional forms of optimal control problems, including time-optimal trajectory generation methods.

3.1.4 Direct and Indirect Solution Methods

There are two main approaches to solving optimization problems in this form: Direct and Indirect optimization methods. Direct methods involve solving for the optimal control directly, while indirect methods solve for the optimal control by establishing a Two-Point Boundary Value Problem using Pontryagin's Maximum Principle and using the resultant co-states to find then find the optimal control variables [11].

3.1.5 Pontryagin's Maximum Principle

Now that the OCP (Optimal Control Problem) has been properly defined, a solution framework will be derived. Pontryagin [9] derived the necessary criterion for solving the OCP outlined in Section 3.1.3 and formulated Pontryagin's Maximum Principle (PMP). PMP allows for finding the equations for the optimal control variable derivatives with respect to time directly. Once the derivative equations are determined, a two-point boundary value problem can be solved numerically for the optimal trajectory. These PMP-based trajectory solutions will serve as the benchmark performance standard along with the reference training data for neural network based controllers.

PMP states that the optimal control takes the form [9] in Equation 3.10:

$$(u^*, \vec{\alpha}^*) = \arg\min_{(u,\vec{\alpha})} H \tag{3.10}$$

"H" in this equation is the Hamiltonian for the optimal control problem and is constructed using costates, which are analogous to Lagrange multipliers. Pontryagin showed that the optimal control for an OCP is the control input that optimizes the Hamiltonian [37]. Therefore, for the system in Eq. 3.1,

the optimal control inputs are the throttle and thrust direction (denoted by u^* and $\boldsymbol{\alpha}^*$) that minimize H. The performance metrics defined in Eqs. 3.7 and 3.9 are cost functions to be minimized over the trajectory; thus, optimizing the Hamiltonian corresponds to a minimization process in this setting.

The Hamiltonian is a scalar equation defined by:

$$H = L + \vec{\lambda}^T \vec{f}(\vec{x}, u, \vec{\alpha}) \tag{3.11}$$

In this equation, L is the incremental cost, also referred to as the Langrangian. It is simply the integrand from the performance metric that is to be optimized in the trajectory. For the minimum fuel use problem defined in Equation 3.7, the incremental cost would just be $u(t)$ since the max thrust and specific impulse are constants in this analysis and can be pulled out of the integral in the objective function. For the minimum energy performance metric in Equation 3.9, L is $u^2(t)$.

The co-state vector is multiplied component-wise to each of the state derivatives and is defined by $\vec{\lambda}$. There is a corresponding co-state for each state variable in the OCP. The co-state vector can be though of as the incremental cost in optimality as a result to changes in the state. The dynamics governing the motion of the spacecraft in Equation 3.1 describe the constraints in the OCP, whereas the costate vectors contain information on the sensitivity of the objective function on changes in the state.

Pontryagin also defines the final necessary condition for computing the co-states, and therefore the optimal control, with the relationship in Equation 3.12 [9]. The combination of Equations 3.10, 3.11, and 3.12 allow for an analytical solution for the complete set of derivatives for the states and co-states of the OCP.

$$\dot{\lambda}_i = -\frac{\partial H}{\partial x_i} \tag{3.12}$$

3.1.5.1 PMP Applied to the Spacecraft Low-Thrust Problem

Applying the equations of motion to the Hamiltonian definition in Equation 3.11 leads to the following Hamiltonian:

$$H = u + \vec{\lambda_r} \cdot \vec{v} + \vec{\lambda_v} \cdot (-\frac{\mu}{r^3}\vec{r} + \frac{T_{max}u}{m}\vec{\alpha}) - \lambda_m \frac{T_{max}u}{I_{sp}g_0} \tag{3.13}$$

The first u in the Hamiltonian comes from substituting the first performance metric from Equation 3.7 in for L. The remaining terms are just the state vector components being multiplied by each of their respective co-states. The position and velocity terms are combined in vector notation for brevity. Equation 3.12 is now applied to this and the co-state derivatives can be computed now to produce the following system of equations [3]:

$$\begin{aligned}
\dot{\vec{\lambda_r}} &= -\frac{\delta H}{\delta \vec{r}} = \frac{\mu}{r^3}\lambda_v - \frac{3\mu\vec{r} \cdot \vec{\lambda_v}}{r^5}\vec{r} \\
\dot{\vec{\lambda_v}} &= -\frac{\delta H}{\delta \vec{v}} = -\lambda_r \\
\dot{\vec{\lambda_m}} &= -\frac{\delta H}{\delta m} = \frac{-T_{max}u}{m^2}\|\vec{\lambda_v}\|
\end{aligned} \tag{3.14}$$

With these derivatives in place, there are now ten equations for each of the five states and co-states. The control variables are still unknown and need to be determined using Equation 3.10. Lawden [38] covers the concept of the primer vector and Prussing [39] shows that the PMP can be used to show this primer vector is the optimal thrusting direction and is defined in Equation 3.15.

$$\vec{\alpha^*} = -\frac{\lambda_v}{\|\lambda_v\|} \tag{3.15}$$

With the primer vector established in Equation 3.15, finding the optimal control simplifies to solving for u that minimizes H. Algebraic manipulation [3] can be used to update the Hamiltonian equation to Equation 3.16 in order to isolate the term proportional to u:

$$H = -\frac{T_{max}u}{I_{sp}g_0}(\lambda_m + I_{sp}g_0\frac{\left\|\vec{\lambda_v}\right\|}{m} - 1) + \vec{\lambda_r}\cdot\vec{v} - \frac{\mu\vec{\lambda_v}\cdot\vec{r}}{r^3} \tag{3.16}$$

Normally, finding the minimum can be done by setting the derivative with respect to u equal to zero and solving for u, but inspection of Equation 3.16 reveals that the Hamiltonian is linear with respect to u. Since this linear relationship exists and u is also constrained between 0 and 1, the optimal control for u becomes the following:

$$u^* = \begin{cases} 1 & \text{if } \rho(t) > 0 \\ 0 & \text{if } \rho(t) < 0 \\ \in [0,1] & \text{if } \rho(t) = 0 \end{cases} \tag{3.17}$$

where $\rho(t)$ is called the "switching function" and is defined as:

$$\rho = \lambda_m + \frac{I_{sp}g_0\|\boldsymbol{\lambda_v}\|}{m} - 1 \tag{3.18}$$

The switching function is the portion of the Hamiltonian that is multiplied by u *and* changes over time (T_{max} , I_{sp}, and g_0 are all constant). So, if ρ is positive, u should be as large as possible to minimize H, and if ρ is negative, then the opposite is desired. Bertrand and Epenoy [40] show that the

switching function only takes on the value of zero at "isolated points" in a given propagation, so u will take the value of either 0 or 1 throughout the trajectory. This is referred to as "bang-bang" control and is a reference to the fact that in this control scheme, the optimal thrust is ramped up to its maximum value the instant ρ is positive and alternatively the thrust should instantly be minimized the moment ρ becomes negative. With these equations for u and $\vec{\alpha}$, the necessary conditions for determining the optimal control are achieved since u and $\vec{\alpha}$ are functions of the states and co-states, and analytical solutions for the time derivatives of the OCP states and co-states are also known.

3.1.6 The Two-Point Boundary Value Problem

With the system of equations defined that describe the time-evolution of the OCP, and objective functions established in subsection 3.1.3, the boundary conditions for each of the control problems of interest can now be considered. In both problems, the spacecraft begins in an initial state defined by r_i, θ_i, $\dot{r_i}$, and $v_{\dot{\theta},i}$. A transformation to polar coordinates is required here to enforce boundary conditions at the start and end of the trajectory. The key difference between the two OCPs under analysis here are the terminal boundary conditions for the trajectory.

For the Two-Body Transfer Problem, the desired terminal condition is that the spacecraft enters a circular orbit defined by r_f. Because of this, $\dot{r_f} = 0$ and $v_{\dot{\theta},f}$ is simply the circular velocity prescribed by orbital mechanics:

$$v_{\dot{\theta},f} = \sqrt{\frac{\mu}{r_f}} \tag{3.19}$$

In the TBT problem, the terminal true anomaly is left unconstrained. To solve this problem, 10 boundary conditions are required for the 10 equations. So far, the initial and final values for r, $\dot{r}$, $\dot{v_\theta}$ are

known. The initial values for θ and m are known, but their final values are unknown. Since the final mass is unconstrained, the final co-state for the mass must be zero [3], which provides an additional boundary condition. The Transversality Condition also states that if a final state is free and the time of flight is fixed, then the terminal co-state for that variable is zero [41]. This gives the final condition necessary to solve this problem, with $\lambda_{\theta,f} = 0$.

Because the problem needs to utilize boundary conditions at the initial and final state of the trajectory, this is a Two-Point Boundary Value Problem (TPBVP). This differs from a standard initial value propagation where a complete set of initial conditions are known along with the state derivatives, and can simply be numerically integrated. In this instance, the initial values of the co-states are all unknown and need to be determined but there is sufficient information at the initial and terminal states to solve the system of equations. For a TPBVP, an iterative numerical process is required to solve the system of equations while satisfying the boundary conditions.

A TPBVP is also necessary to solve the Two-Body Rendezvous problem, but the boundary conditions used to solve it are different. In the TBR problem, the final position and velocity conditions: r_f, θ_f, $\dot{r_f}$, and $v_{\theta,f}$, are all fixed. Again, for mass optimality, the final co-state for mass is zero, which provides all the necessary conditions for solving the TBR problem.

3.1.7 Homotopic Smoothing

In section 3.1.5.1, the notion of "bang-bang" control was discussed, and it is evident that this control regime is what leads to fuel-optimal control. The thrust is applied either at maximum capacity, or not at all. This discontinuous control regime is problematic, particularly for the root finding algorithms necessary to solve the TPBVP as they prefer to operate with smooth functions to reach

convergence.

To counteract this, Homotopic Smoothing to first solve for the "energy-optimal" cost function (which does not have the "bang-bang" control characteristic) and then gradually move from this energy-optimal trajectory to a mass-optimal one via this smoothing process. Bertrand and Epenoy [40] cover a variety of potential smoothing methods that can be used to address this issue, but the energy-fuel continuation process used in this analysis is given by adjusting the spacecraft throttle as follows [42]:

$$u^{\star}(\rho,\epsilon) = \begin{cases} 0, & \rho > \epsilon, \\ 1, & \rho < -\epsilon, \\ \frac{1}{2}\left(1 - \frac{\rho}{\epsilon}\right), & |\rho| \leq \epsilon, \end{cases} \qquad \epsilon > 0. \tag{3.20}$$

In this control scheme, the same switching function ρ from Equation 3.18 is used, but now there is a smoothing parameter ϵ that dictates the amount of smoothing to take place. This can be thought of as a weighting between using the energy cost index and the fuel cost index, with $\epsilon = 1$ translating to a pure energy-optimal trajectory, and $\epsilon = 0$ a completely fuel-optimal trajectory. Notice that when ϵ is at or near zero, the likelihood that the switching function is in the smoothed regime in the third column is low, and the control effectively becomes "bang-bang", as expected for fuel-optimal trajectories. In this analysis, epsilon begins at a value of one, and is reduced in each iteration by a reduction factor γ_h:

$$\epsilon_{k+1} = \epsilon_k \gamma_h \tag{3.21}$$

When $\epsilon = 1$, the radius of convergence for the problem is much higher, and the numerical issues

related to the sharp discontinuities are less of a factor in the targeting process. Once an initial co-state vector that satisfies the TPBVP is found, ϵ can be gradually reduced for a fuel-optimal trajectory. Figure 4.3 shows how the thrust profile evolves with the reduction in the smoothing parameter ϵ for the same boundary conditions in the TPBVP. When $\epsilon = 1$, the thrust is very smooth and gradually responsive to the switching function and root-finding algorithms converge with much higher rates of success. As ϵ is reduced, the thrust profile evolves much more rapidly in time. From empirical testing, an $\epsilon = 10^{-4}$ is typically sufficient to accurately replicate "bang-bang" control.

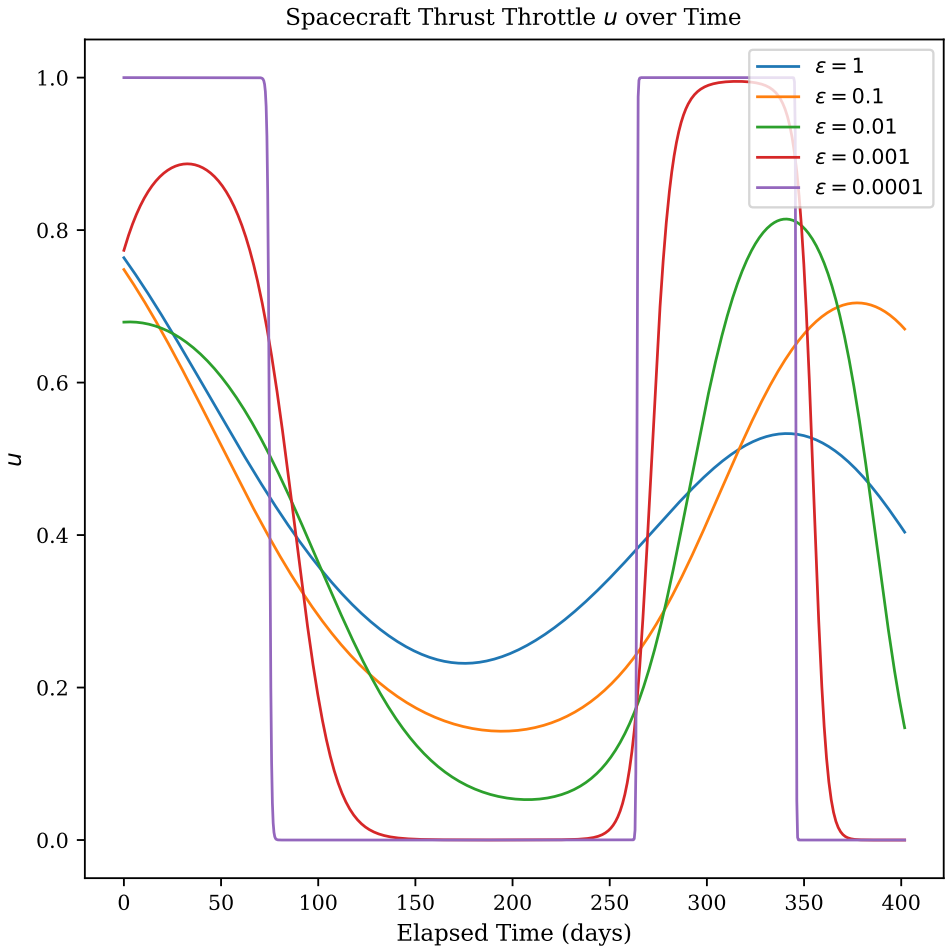
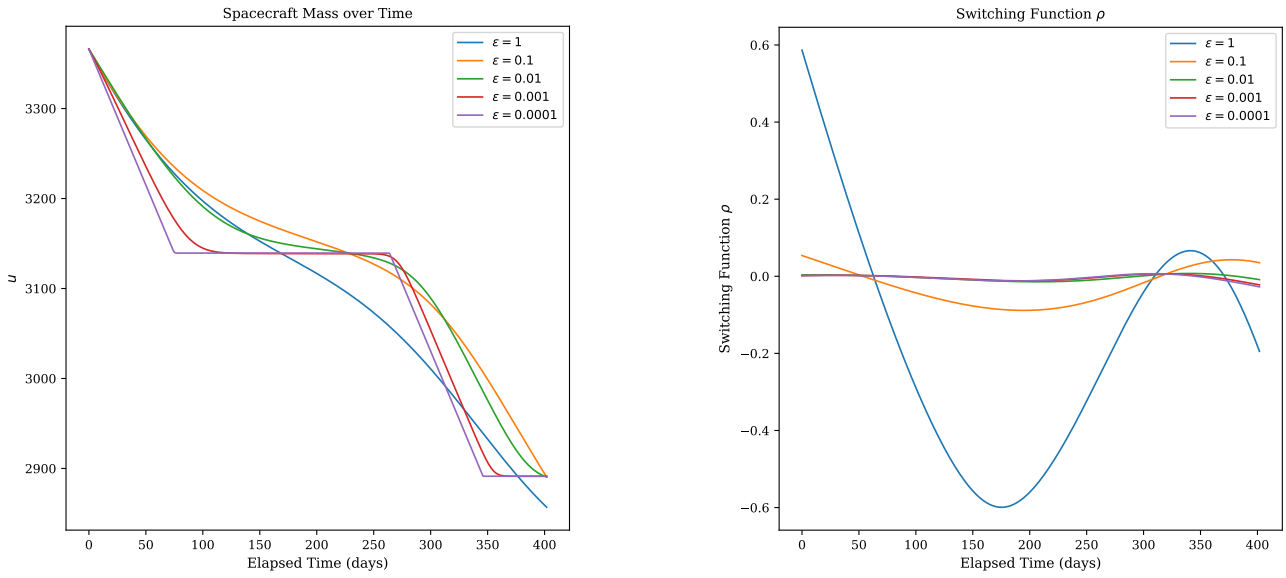


Figure 3.1: Progression of Spacecraft Thrust Throttle Control with Homotopic Smoothing



(a) Throttle

(b) Switching Function

Figure 3.2: Smoothing progression of spacecraft mass use and switching function.

3.2 Reinforcement Learning

At this point, the methods for generating the a set of mass-optimal trajectories has been covered. The use of these trajectories for supervised "pre-training" will be covered later in this work. Reinforcement learning will be used as the baseline approach for applying machine learning to the TBT and TBR problems.

Reinforcement learning (RL) is a subfield of machine learning that is focused on optimizing the actions of an agent in a simulated environment. The agent in the environment is the decision maker in this approach, and refines estimates for what actions achieve desired outcomes in the environment. The environment is the framework that the agent operates in, and contains the information about the dynamics of the problem being solved. The agent receives information by performing actions in the environment and learning from the resulting evolution in its state. The diagram in Figure 3.3 is typical in RL literature and depicts the information feedback loop in this relationship:

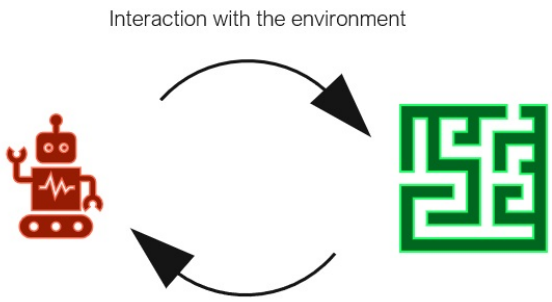


Figure 3.3: Reinforcement Learning Feedback Loop

Reinforcement learning has been applied to problems in various disciplines spanning engineering, finance, game theory, mathematics and more. RL blends exploiting knowledge gained by an agent navigating its environment to maximize performance and the systematic exploration of new actions to prevent settling into suboptimal behaviors. RL differs from the optimization processes discussed earlier because there is no explicit knowledge of the dynamics governing the system. It also stands apart form supervised learning techniques because it does not rely on labeled data for training either. There are no reference trajectories to compare to (at least in the standard RL approach), the agent must learn by trial and error.

To understand the dynamic programming techniques that can be applied to problems in RL, and the mechanics by which an agent learns in this ML paradigm, an explanation of Markov Decision Processes is required. These types of processes are the mathematical containers that capture the necessary components needed for using RL solution techniques.

3.2.1 The Markov Decision Process

To use RL, the problem under consideration must be formulated as a Markov Decision Process or MDP. A MDP framework is an encapsulation fo the dynamics of the system being analyzed. MDPs

bundle the information of the problem into four major categories: the state space, the action space, the rewards, and the transition functions.

The state vector contains all the relevant information about the agent/environment at a specific point in time and lies within the state space, S. The action space A is the set of all allowable actions an agent can take while navigating the environment. Within the MDP structure, a reward space R is the set of rewards gained by the agent when performing an action in a state. Finally, the transition space T is the set of transition functions that describes the distribution of possible states an agent can move into s_{i+1} given an action a_i in the state s_i. In addition to these spaces, a discount factor, γ, is included in the MDP formulation. The discount factor is a multiplier applied to the reward at the current timestep in the MDP, and it discourages an agent from pursuing rewards over an infinite time horizon. The goal of the agent in RL is to maximize the *expectation of the discounted sum of rewards* in the MDP. The discounted sum of rewards is called the return [1], and is defined by Equation 3.22 for an infinite horizon problem.

$$G_t = \sum_{t=1}^{\infty} r_t \gamma^t \tag{3.22}$$

The discount factor lies between 0 and 1, the smaller the discount factor, the quicker future rewards diminish compared to rewards at the current timestep. Because $\gamma < 1$, the value function is bounded even as the number of time steps approaches infinity.

Figure 3.4 shows the interaction of the four major components of the MDP. The agent starts off at a state $s_i \in S$ at $t = t_i$. The agent can choose any allowable action in the action space at this time step, denoted by $a_i \in A$. The transition function definitions for the environment then determine what new state the agent transitions to after performing the action a_i in the state s_i at time t_i and the reward

function definition provides the reward for arriving at the updated state:

$$s_{i+1}, r_i \leftarrow T(s_i, a_i) \tag{3.23}$$

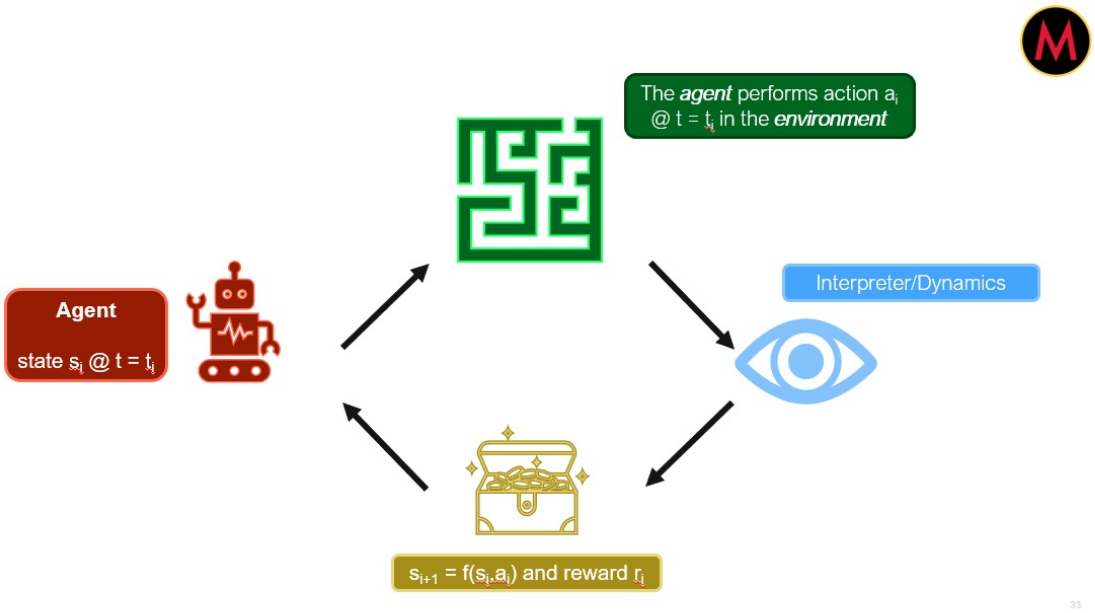


Figure 3.4: The Markov Decision Process

The Markov property is a crucial element of a MDP and is a required condition for RL methods to be applied to this framework. A stochastic process is said to have the Markov property if future states are dependent only on the current state and not the *history* of states [43]. A Markovian process can be considered memoryless, and relies only on the information available in the current state. The transition functions in the MDP must adhere to this principle, the next step must be a function of just the current state and action, and not a history of states and actions. The motion of a spacecraft as described in this analysis is a Markovian process because the equations of motion are all derivatives that can be computed with the spacecraft's current position and velocity, which are assumed to be known with perfect knowledge here.

3.2.2 State and Action Value Functions

The goal of an agent is to perform actions in states so that its discounted reward is maximized, as described in section 3.2.1. The concept of a *policy* is introduced here to assist in understanding how agents can optimize their behaviors in RL. A *policy*, often denoted by $\pi(s)$ in RL literature, is a mapping of an agent's state to an action that is performed in that state. Depending on the framing of the problem, the policy can return directly return the action given a state as input, $a \leftarrow \Pi(s)$, or the probability of performing a certain action given the current state: $p_a \leftarrow \pi(a|s)$. A given policy is considered to be the optimal policy, if it satisfies the following condition [1]:

$$\pi^*(s) = \arg\max_{\pi} U^{\pi}(s) \tag{3.24}$$

In this equation, $U(s)$ is known as the state value function, and it quantifies the value of the agent being in a particular state. There is an alternative flavor of the state value function, the action value function $Q(s, a)$, and this more specifically quantifies the value of an agent performing a particular action in a particular state. These quantities are defined by the following:

$$\begin{aligned} U^{\pi}(s) &= \mathbb{E}_{\pi}\left[G_t \mid s_t = s\right], \\ Q^{\pi}(s, a) &= \mathbb{E}_{\pi}\left[G_t \mid s_t = s,\ a_t = a\right]. \end{aligned} \tag{3.25}$$

The expectation operator, $\mathbb{E}$, is used to account for the fact that MDPs are often stochastic, and the expectation operation is required to determine the mean return of all possible outcomes. These value functions are central to RL solution algorithms, as they measure the long-term expected rewards for the MDP. Going one step further, these value functions can be written in a recursive form by

separating out the immediate reward and then considering the value of the next state. This recursive form is called the Bellman Expectation Equation or BEE and is defined in the state-value and action-value flavors below [44]:

$$\begin{aligned} U^{\pi}(s) &= \mathbb{E}_{\pi}[\, r_t + \gamma V^{\pi}(s_{t+1}) \,|\, s_t = s] \,, \\ Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[\, r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) \,|\, s_t = s,\ a_t = a] \,. \end{aligned} \tag{3.26}$$

These equations are important because they relate the value function to itself one step in the future, and it is this recursive property that is leveraged in most of the RL algorithms for optimizing an agent's policy.

3.2.3 Policy Optimization Algorithms

There are several classes of algorithms for computing optimal policies in RL. Value-based methods focus on learning a value function (either $U^{\pi}(s)$ or $Q^{\pi}(s,a)$) from which a control policy is obtained by acting greedily with respect to these estimates [43]. These methods typically involve storing these estimates in tabular data structures and are used often in discrete action spaces. Policy gradient methods optimize the set of parameters θ of a policy function $\pi_{\theta}(a|s)$ such that a performance index is optimized. In this instance, the performance index is the value function given the policy defined earlier. Gradient ascent is performed to adjust the parameters of the policy function in order to maximize the performance metric [45] shown in Equation 3.27:

$$J(\theta) = \mathbb{E}_{\pi_{\theta}}[G_0] = \mathbb{E}_{\pi_{\theta}}\left[\sum_{t=0}^{\infty} r_t \gamma^t\right], \tag{3.27}$$

In practice, it is more effective to combine policy gradients, with a learned value function. This

practice leads to a class of RL solutions called "actor-critic" methods [46]. In this framework, the "actor" is a the policy approximation $\pi_\theta(a|s)$, and the "critic" is a learned value approximation: U_ϕ or Q_ϕ. The ϕ subscript notation for the value function approximation refers to the critic parameters being learned in this RL process, and a different symbol is used to differentiate it from the actor parameters for the policy θ; however the both just represent the distinct set of parameters belonging to the actor and critic respectively. The motivation for using actor-critic methods lies in its smoother learning process. The critic learns to predict the value of states or state-action pairs, and the actor uses this value prediction, rather than the raw rewards/returns themselves. These value predictions are smoother and less noisy than the rewards themselves, and thus the actor works with a clearer learning signal [43].

Since actor-critic methods learn from experience generated by a policy, it is important to distinguish how the agent in RL gathers this experience. There are two main mechanisms for experience gathering: on-policy and off-policy methods. In on-policy RL, the agent learns from data *only* generated by the current policy. This is often a simpler learning process with less variance; however, the trade-off here is that on-policy learning is less efficient at finding the global optimal. Off-policy learning, on the other hand, learns from data generated by any policy (can be previous policy iterations or the current policy). This is more efficient at finding the optimal policy but the drawbacks are that added complexity is introduced in the learning process and more noise has to be sorted through as a result of all off-policy data.

In summary, there are several categories of policy optimization algorithms: value-based methods, policy gradient algorithms, actor-critic methods, and on-policy and off-policy learning. These methods reference the policy and/or value networks generically as "functions" or "approximations". The following section covers what types of functions and data structures work best for the problems

of interest in this work.

3.3 Deep Reinforcement Learning

For discrete problems, the functions that represent the value or policy functions can be as simple as look-up tables containing the values of the allowable actions or reachable states. Consider the simple hex-world problem below from Chapter 7 in Kochenderfer [1] in Figure 3.5.

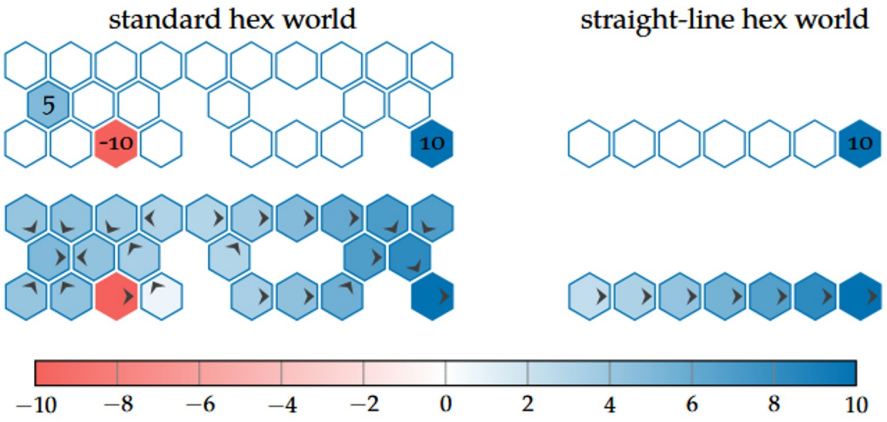


Figure 3.5: The Hex-World Problem [1]

In this toy problem, an agent must navigate the environment until one of the terminal states are reached (the hexes with the reward values of -10, 5, and 10). There is a small chance that the agent randomly moves in a unintended direction in this problem to add stochasticity to the process. The individual values for each of the hex states can be computed directly by recursively applying BEE (Equation 3.26) in just a couple of iterations. In this simple example, the number of states to account for is small and so these tabular methods can be used without issue. An obvious problem arises when attempting to apply these methods to continuous or even moderately large discrete problems: the compute and memory required to reason over the entire state and action space becomes intractable.

For larger/continuous problems, functional models are required to approximate the policy or value networks in the RL approaches of Section 3.2.3. There are several classes of function approximators shown in the Figure 3.6 below:

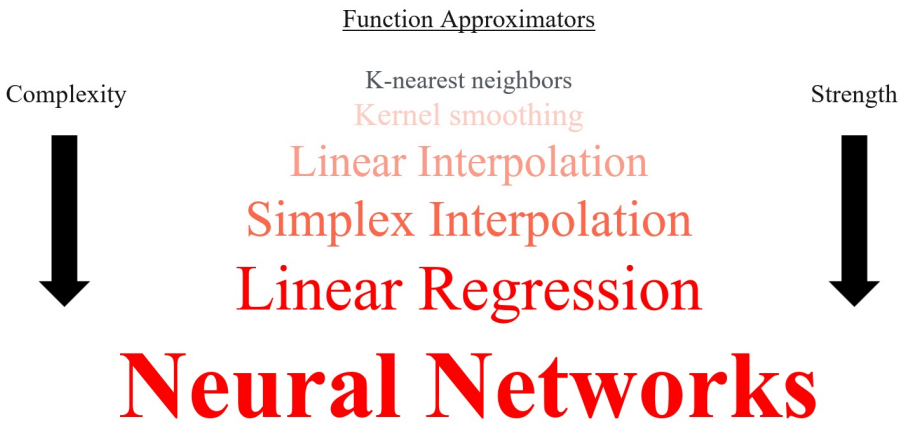


Figure 3.6: Function Approximators

These function approximation methods range in sophistication and capability, but neural networks on the more sophisticated end of the spectrum, show the most promise for use in RL. The term "Deep Reinforcement Learning" (DRL) refers to the class of reinforcement learning techniques described in 3.2.3, but with the specific use of artificial neural networks to approximate the agent's policy and value functions. The term "Deep Learning" refers to a type of machine learning that uses neural networks containing multiple layers, and so DRL uses these multi-layered networks in the learning process for their ability to learn more complicated functions.

3.4 Neural Networks

Neural networks act as function approximations and can be trained to reproduce this desired mapping of inputs and outputs and are inspired by the neurons in organisms. They are particularly useful when the underlying model of a process of interest is unknown or expensive to compute accurately. A neural network is really a collection of individual neurons connected in some sort of configuration, and each of these neurons has a set of parameters associated with it. These parameters are analogous to coefficients in a polynomial and can be adjusted iteratively to produce a desired output in the training process. The neurons in neural networks are typically chained together in layers, with the outputs of one layer serving as the inputs for the subsequent layer. The figure below shows a representation of an individual neuron:

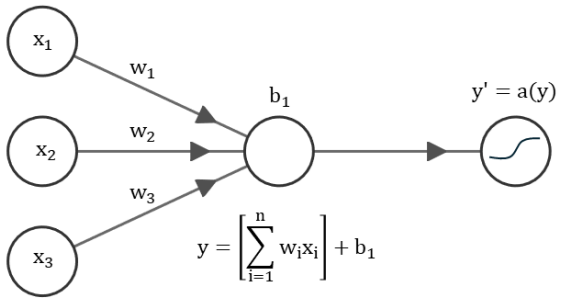


Figure 3.7: Neuron Representation

There are three main mathematical processes involved in the propagation of information in a neuron. The first operation involves weighing the input signals by weight parameters associated with the neuron itself. The second operation is to take the sum of all these weighted inputs and apply a bias as illustrated in Figure 3.7. Up to this point, this is still a linear model. Most interesting behavior that neural networks can be applied to approximating is non-linear, so to introduce some non-linearity, an

activation function is applied to the preliminary output from the prior step. This is done to increase the complexity of behavior that neural networks can predict. The selection of the activation function is design decision and often depends on the situation, but common examples involve the hyperbolic tangent function, the sigmoid function, or the rectified linear unit.

This analysis will use neural networks to approximate the optimal low-thrust control vector of a spacecraft given its current state. The neural network in 3.8 depicts this goal, with the current spacecraft state serving as the input to the neural network on the left, with the output, in this instance the mass optimal control, being output on the right.

The starting baseline for supervised learning in this analysis will be drawn from Rubinsztejn, Sood, and Laipert [6] for initializing a neural network low thrust controller. The authors show that this neural network controller can replicate near-optimal control for transfers from Mars to Earth orbit, motivating its use as a starting point in this analysis. This controller does suffer from the limitations discussed previously in that it is trained for a specific type of trajectory (Mars to Earth transfers) for a specific spacecraft hardware configuration (i.e. the max thrust, propulsion system specific impulse, spacecraft mass, etc., are implicitly assumed to be constant by the network). The network ingests the spacecraft's current state, consisting of a 5-element state vector and returns the resultant control vector in a 3-element output vector. A sample schematic for the network is provided in Fig. 3.8. Note that this network will require an update when target-generalized training takes place to include the target state vector as an input feature of the network.

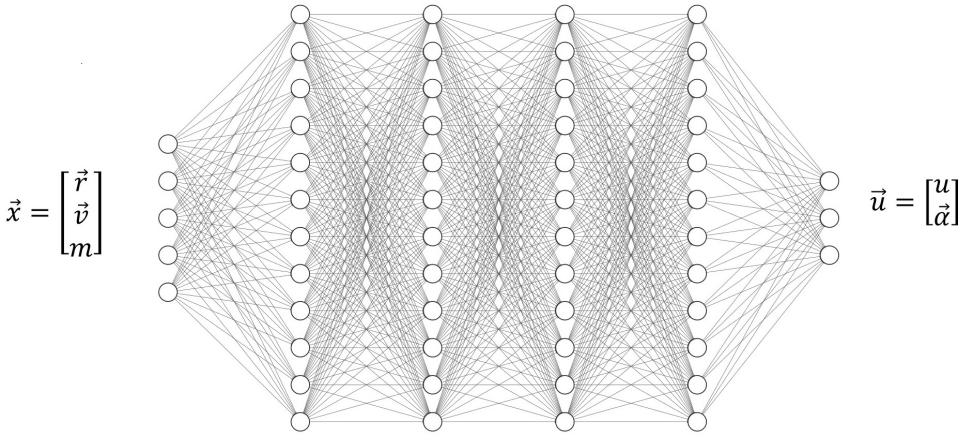


Figure 3.8: Neural Network Controller Diagram (Not Drawn to Scale)

3.5 Supervised Learning

The process for approximating the optimal control of a spacecraft using supervised deep learning will now be discussed. At its core, supervised deep learning is a type of machine learning that uses a combination of neural networks and labeled data to approximate a mapping of inputs to outputs. This approach relies on having a dataset prior to training a model (whether it be simulated or collected empirically from the world, etc.) that contains the inputs to the model along with the corresponding outputs. The inputs are also referred to as features, and these are the data that are expected to be known outside of the training environment. The outputs are the desired mapping based on the training data provided.

3.6 Deep RL Algorithms and the Soft Actor-Critic

3.6.1 A Brief Landscape of Deep Reinforcement Learning Algorithms

Deep reinforcement learning (DRL) refers to reinforcement learning methods that use deep neural networks as function approximators for the policies and value functions as discussed in Section 3.3. Modern DRL algorithms are often categorized in several subtypes: value-based versus policy-based learning, on-policy versus off-policy data usage, and deterministic versus stochastic policies. In practice, continuous-control problems such as spacecraft trajectory control tend to favor actor-critic methods that can represent continuous actions directly. A taxonomy for RL algorithms is provided in Figure 3.9.

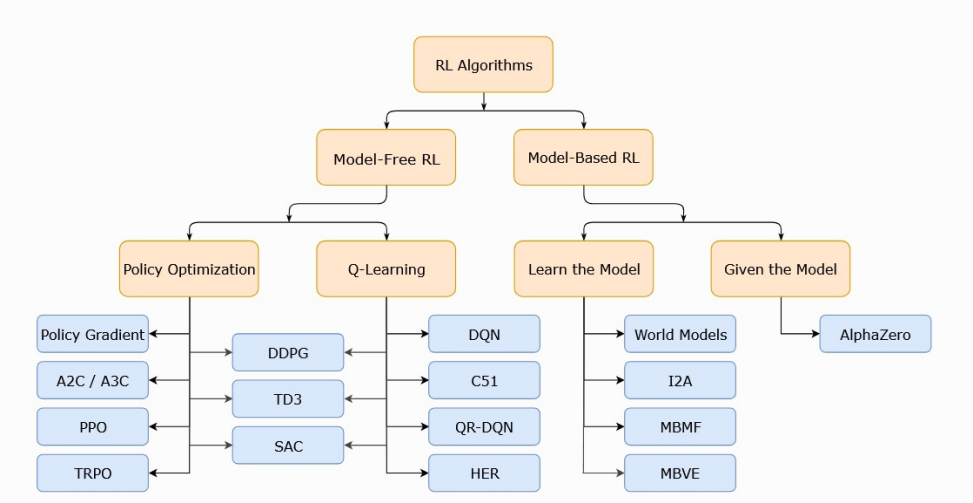


Figure 3.9: Deep Reinforcement Learning Algorithm Taxonomy [2]

An important distinction in DRL is whether learning is *on-policy* or *off-policy*. On-policy algorithms like A2C/A3C, and PPO update the policy using data generated only by the current policy, which can lead to stable learning but often at the cost of sample efficiency. Off-policy methods (e.g., DDPG, TD3, SAC) can reuse data collected by previous policies via a replay buffer, improving sample

efficiency but introducing additional algorithmic complexity.

For continuous control applications like the low-thrust problem, widely used algorithms include deterministic actor-critic methods such as DDPG and TD3, and stochastic actor-critic methods such as Soft Actor-Critic (SAC). SAC is particularly attractive when exploration is challenging and simulator interactions are expensive, because it is a off-policy algorithm and explicitly encourages exploration through entropy maximization [32].

3.6.2 Soft Actor-Critic

The Soft Actor-Critic algorithm is an off-policy, model-free actor-critic reinforcement learning approach designed for continuous action spaces and is a desirable approach when state and action spaces are highly dimensional. SAC augments the standard RL objective with an entropy term that encourages stochasticity in the policy. This produces two practical benefits: improved exploration during training and a natural trade between exploration and exploitation governed by a temperature parameter.

Let $\pi_\theta(a \mid s)$ denote a stochastic policy (actor) with parameters θ, and let $Q_\phi(s, a)$ denote a critic with parameters ϕ. SAC optimizes something different that standard RL algorithms by utilizing a maximum-entropy objective of the following form [32]:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha\, \mathcal{H}(\pi(\cdot \mid s_t))\right)\right], \tag{3.28}$$

where $\mathcal{H}(\pi(\cdot \mid s)) = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log \pi(a \mid s)]$ is the policy entropy and $\alpha > 0$ is the temperature parameter [32]. Larger α weights exploration more heavily by rewarding higher-entropy policies, while smaller α promotes exploitation over exploration.

In SAC, the critic is trained using a soft Bellman backup, meaning the target value includes both the expected future return and an entropy term that encourages exploration [**?**]. Many implementations use two critic networks to reduce overestimation bias by forming targets using the more conservative of the two value predictions [47]. To improve stability, these targets are computed using slowly updated target networks rather than the most recent critic parameters. For a transition (s_t, a_t, r_t, s_{t+1}), the resulting soft target can be written as

$$y_t = r_t + \gamma\, \mathbb{E}_{a_{t+1}\sim\pi_\theta(\cdot|s_{t+1})} \left[\min_{i\in\{1,2\}} Q_{\bar{\phi}_i}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\theta(a_{t+1} \mid s_{t+1}) \right]. \tag{3.29}$$

and each critic is trained by minimizing a mean-squared Bellman error,

$$J_Q(\phi_i) = \mathbb{E}\left[\left(Q_{\phi_i}(s_t, a_t) - y_t \right)^2 \right], \qquad i \in \{1, 2\}. \tag{3.30}$$

The actor is updated to prefer actions that are predicted to have high value under the critic while remaining sufficiently stochastic (the return and entropy components in the SAC objective from Equation 3.28). This trade between exploitation and exploration is controlled by the temperature parameter α. A common form of the policy objective is

$$J_\pi(\theta) = \mathbb{E}_{s_t} \left[\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)} \left[\alpha \log \pi_\theta(a_t \mid s_t) - \min_{i\in\{1,2\}} Q_{\phi_i}(s_t, a_t) \right] \right], \tag{3.31}$$

with actions sampled from a parameterized distribution (often a squashed Gaussian) to enable efficient gradient-based learning [32].

Finally, many SAC implementations tune α automatically to maintain a desired level of policy entropy, which reduces manual hyperparameter selection and helps keep exploration consistent

Algorithm 1 The Soft Actor-Critic [32]

Initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$.
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\phi(a_t \mid s_t)$
    $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \widehat{\nabla}_\psi J_V(\psi)$
    for $i \in \{1, 2\}$ **do**
      $\theta_i \leftarrow \theta_i - \lambda_Q \widehat{\nabla}_{\theta_i} J_Q(\theta_i)$
    end for
    $\phi \leftarrow \phi - \lambda_\pi \widehat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for

throughout training:

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_\theta(\cdot|s_t)} \left[-\alpha \left(\log \pi_\theta(a_t \mid s_t) + \mathcal{H}_{\text{tgt}} \right) \right]. \tag{3.32}$$

With these components defined, we can examine a high-level algorithm that illustrates the SAC approach provided by Haarnoja et al. [32] in Algorithm 1. There are two main processes here, an exploration step and a network updating step. In the top exploration loop, the agent samples actions from the policy network and steps through the environment. The resultant transitions and experiences are added to the experience replay buffer. In the second main process, gradient based updates are performed on the SAC networks (the actor, twin critics, and value networks) to improve the policy and value estimates with the new experiences. Typically the user can specify the amount of environment exploration steps to perform and gradient updates to perform for a given iteration, in this work, an iteration will consist of one gradient update per environment step.

Overall, SAC combines a stochastic actor with critic-based value learning, stabilized by target networks and improved sample efficiency through off-policy updates using a replay buffer. The

following section describes the replay buffer and its role in off-policy DRL.

3.6.3 The Replay Buffer

Off-policy algorithms such as SAC typically store experience in a replay buffer $\mathcal{D}$ containing experience tuples:

$$e_t = (s_t, a_t, r_t, s_{t+1}, d_t), \tag{3.33}$$

with s_t, a_t, r_t, and s_{t+1} being the MDP components discussed previously along with a terminal boolean indicator d_t denoting if the achieved state is a terminal state. During training, mini-batches are sampled uniformly at random from $\mathcal{D}$ to compute stochastic gradient updates for the actor and critic networks. Randomized sampling breaks the strong temporal correlations present in sequential rollouts and enables repeated reuse of past experience, improving data efficiency relative to on-policy methods [32, 48]. In this work, the replay buffer stores state transitions generated by the spacecraft simulation under the current and prior policies, and SAC performs updates using these sampled mini-batches while the environment continues to generate new experience.

Chapter 4: Methodology

4.1 Training Data Generation

The process for generating the datasets using homotopoc smoothing will now be covered. The majority of the algorithmic process used in this work for generating mass-optimal trajectories using indirect methods was drawn from Sidhoum and Oguri [3], and Figure 4.1 summarizes the algorithmic process used nicely. First, the initial conditions are specified for the targeting algorithm, with the initial position and velocity of the spacecraft, the time of flight, and the target location known *a-priori*. Recall that the boundary conditions differ depending on the problem but the same iterative process can be applied to both the TBT and the TBR. Some new parameters are introduced in this optimization process. Beginning with the input block, a new input parameter γ must be supplied to this version of the targeter, and this dictates the rate of smoothing in this process.

The single shooting method is employed here to solve the TPBVP steps in this process. A simplified one-dimensional example of the single shooting method is shown in Figure 4.2. The initial costate guesses are varied at the beginning of the shooting process. This initial co–state guess is paired with the known state vector components and propagated forward to the final set of known boundary conditions. The deltas between the achieved final conditions and the known final conditions are computed and used to inform the next co-state guesses. This process is repeated until the differences between the final known boundary conditions and the achieved boundary conditions are sufficiently small. A final propagation with no smoothing is performed to generate a bang-bang fuel optimal trajectory that satisfies the two sets of boundary conditions.

The iterative process employed here involves the shooting method described in subsection 3.1.7

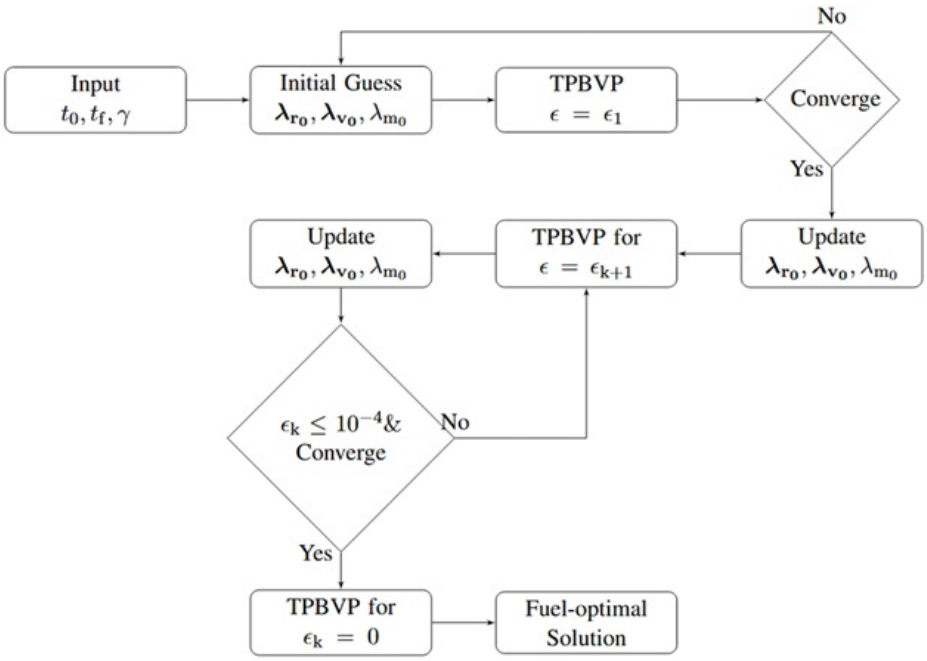


Figure 4.1: Block Diagram for Solving the Two-Point Boundary Value Problem with Smoothing (Sidhoum and Oguri) [3]

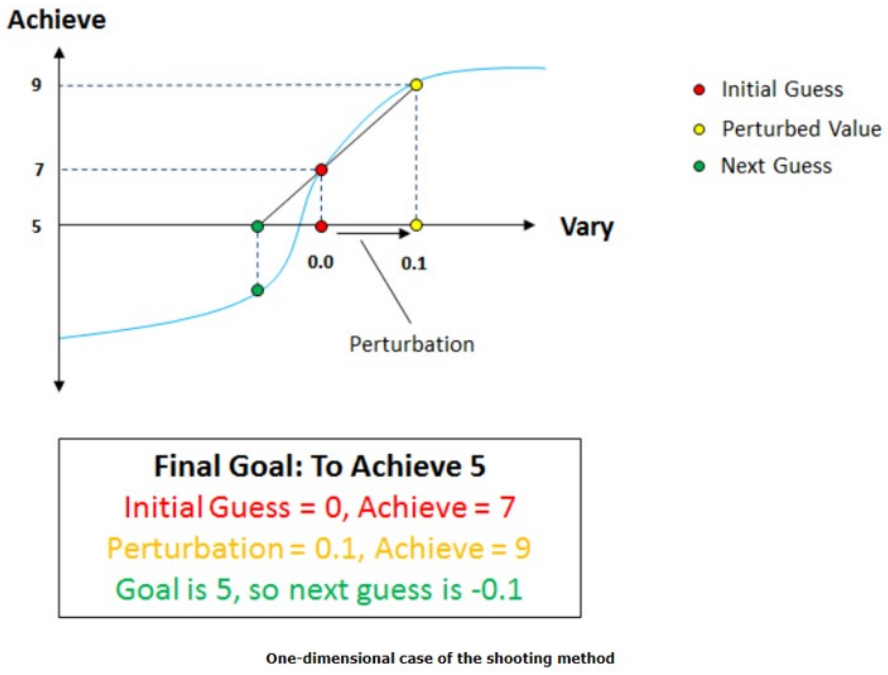


Figure 4.2: One-Dimensional Single Shooting Method Example [4]

to repeatedly solve the TPBVP. This smoothing process is conducted to progressively move from a suboptimal trajectory that has an initially high convergence radius, to a truly mass-optimal trajectory with a much smaller convergence radius. This process is illustrated in Figure 4.3, where the sample trajectory is iteratively smoothed until mass optimality ($\epsilon \rightarrow 0$) is achieved for the same initial and final target conditions. Each thrust profile plotted is for the same boundary conditions, and the only changing variable is ϵ. The control output vector consists of the spacecraft thrust throttle u (the fraction of available thrust to maneuver at) and components of the unit vector of the thrust direction, denoted as α_x and α_y, however, only u is shown in Figure 4.3.

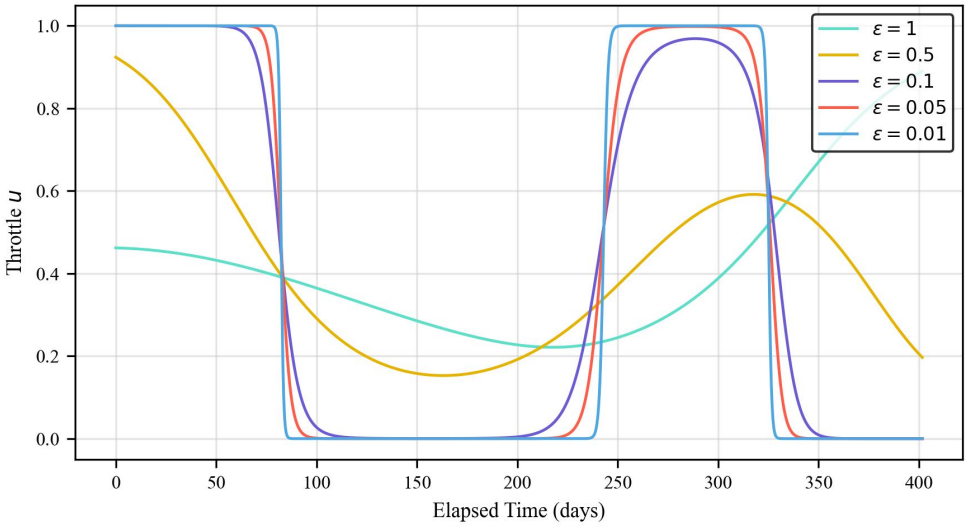


Figure 4.3: Progression of Spacecraft Thrust Throttle Control with Homotopic Smoothing

These optimal profiles can be computationally expensive to generate due to this iterative smoothing process that is required to solve for fuel-optimal control. A high-performance compute (HPC) cluster was used to generate large amounts of training data in parallel. Each trajectory is saved to an ephemeris file with 100 state vectors evenly distributed through time for a given trajectory. These ephemeris files are then saved for the trajectory and used for the pretraining process. There are two main experiments conducted in this work to assess the effects with pretraining on fuel optimal trajec-

tories, and a specific dataset is used for each of these experiments. The first experiment uses a smaller and relatively simple dataset containing transfers between circular orbits between the SMA ranges of the Earth and Mars. The second dataset is larger and involves more complicated transfers that involve variations in eccentricity and the argument of periapsis in the initial and final orbits.

4.1.1 Circular Transfer Dataset

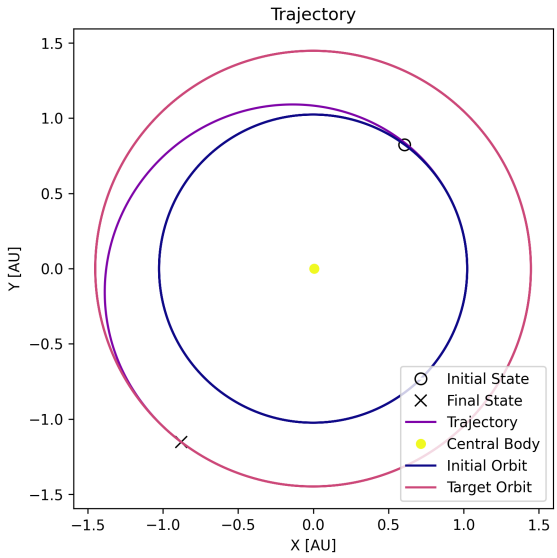
Table 4.1 contains the relevant parameters used to generate the initial circular transfer dataset for the TBT problem. The spacecraft hardware parameters I_{sp}, T_{max}, and initial mass, m_0, are held constant in this analysis, and their values are drawn from the Earth Return Orbiter spacecraft in the Mars Sample Return mission [49]. For variables in the table that have a range, the values are sampled uniformly between the listed bounds for each trajectory solution.

Table 4.1: Circular Transfer Training Data Parameters

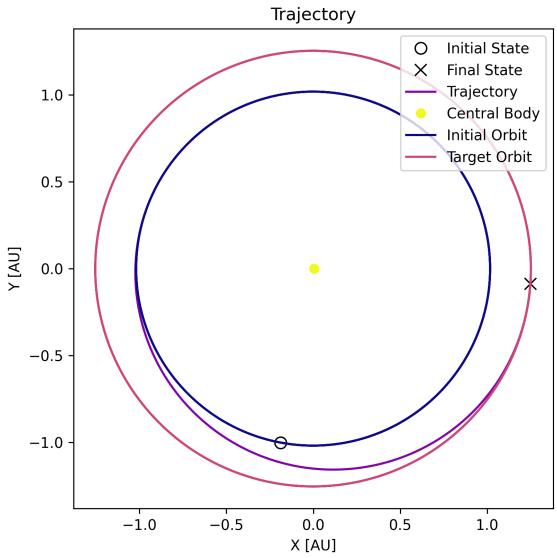
Variable	Type	Description	Value
TOF	Input	Time of Flight	1–1.9 years
I_{sp}	Input	Specific Impulse	3872 s
$T_{\max}$	Input	Max Thrust	1.33 N
μ_{sun}	Input	Central Body Gravitational Parameter	$1.3 \times 10^{11}\,\text{km}^3/\text{s}^2$
m_0	Input	Initial Spacecraft Mass	3366 kg
a_{initial}	Input	Initial orbit SMA	1–1.52 AU
e_{initial}	Input	Initial orbit eccentricity	0.0001 (fixed)
ω_{initial}	Input	Initial orbit argument of periapsis	0 deg (fixed)
ν_{initial}	Input	Initial orbit true anomaly	0–360 deg
a_{final}	Input	Final orbit SMA	1–1.52 AU
e_{final}	Input	Final orbit eccentricity	0.0001 (fixed)
ω_{final}	Input	Final orbit argument of periapsis	0 deg (fixed)
ϵ_i	Internal	Initial homotopic smoothing parameter	1
ϵ_f	Internal	Final smoothing tolerance	10^{-4}
γ	Internal	Smoothing rate	0.5
N_{traj}	Output	Number of trajectories generated	1,000
N_{vectors}	Output	Number of vectors per file	100

The ϵ parameter measures the degree of smoothing, with $\epsilon = 1$ being the maximally smoothed trajectory, and $\epsilon = 0$ denoting the "bang-bang" thrust profile. The initial epsilon is multiped by the smoothing rate, γ, after each iteration until e_k is less than the final smoothing tolerance.

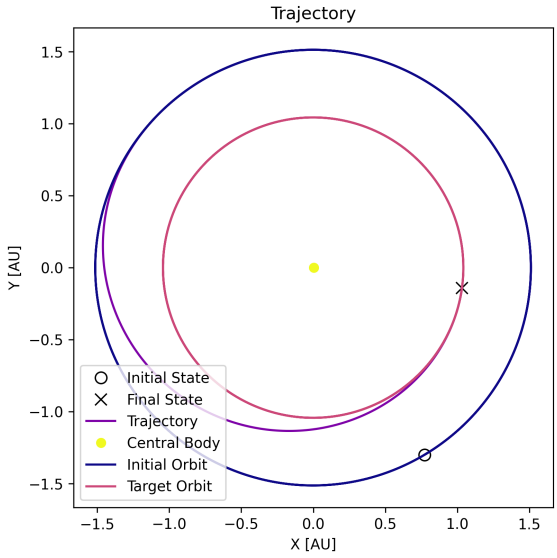
The indirect optimization process was used to generate 1,000 transfers to and from these circular orbits between 1 and 1.9 AU (between the SMAs of Earth and Mars). Samples plots of these mass optimal trajectories are shown in Figure 4.4. The distribution of initial and target states for the training trajectories was chosen to match the resetting behavior of the TBT Markov Decision Process. In Figure 4.4, the initial orbits are shown in blue, the target orbits are in pink, and the mass-optimal transfers are shown in purple. The spacecraft is permitted to enter the target orbit along any value of true anomaly. Since both initial and target orbit SMAs in this dataset are uniformly random between 1 and 1.9 AU, there is variation in the spacecraft moving inward or outward for the transfer arcs.



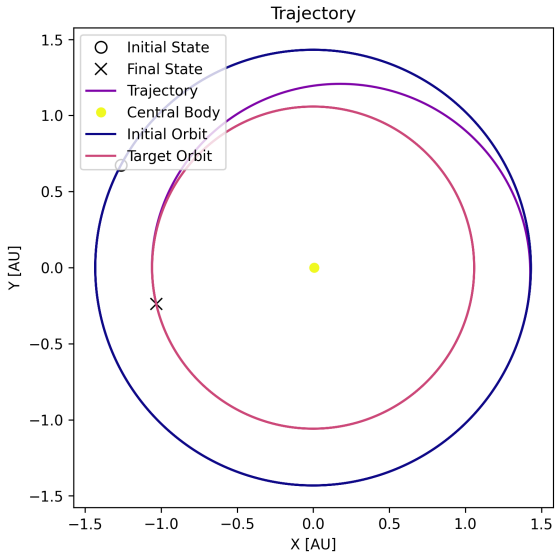
(a) Case 1



(b) Case 2



(c) Case 3

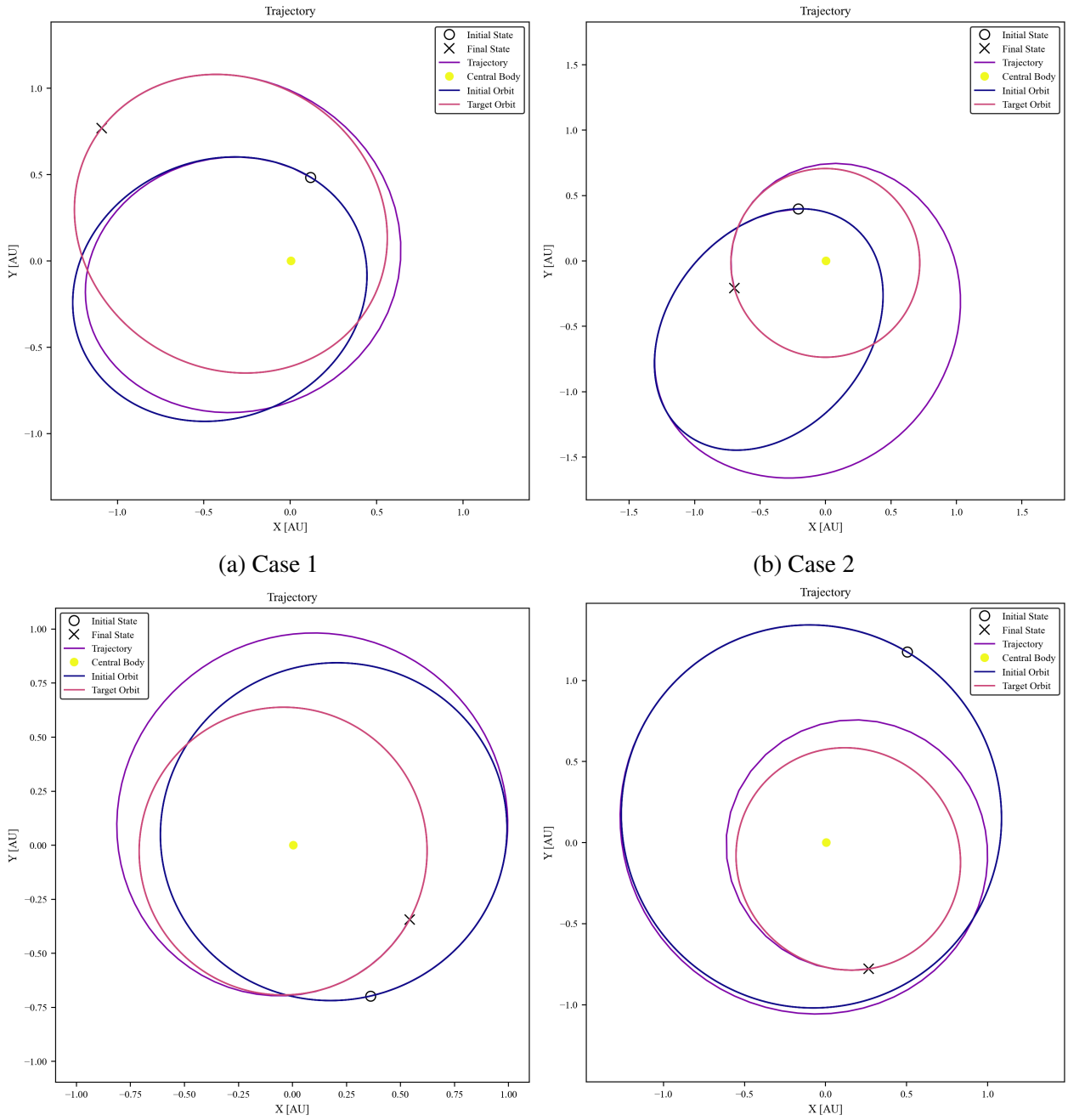


(d) Case 4

Figure 4.4: Sample Trajectories from Circular Transfer Training Data Set

4.1.2 Hard Transfer Dataset

A second dataset was generating using a more expansive set of initial and target orbital element ranges. The eccentricity range was expanded to include more difficult eccentric transfers in the training data, and the argument of periapsis range was expanded to include apse line rotations in the training data, and the semi-major axis range was also increased to include SMAs between Mercury's and Mars' orbits: 0.39 AU and 1.52 AU. This dataset will be referred to the "Hard Transfer Dataset" to distinguish it from the previous simpler training dataset. There are also more ephemerides included in this dataset, ten thousand trajectories versus the one thousand trajectories in the circular transfer dataset. Table 4.2 contains a summary of all the parameters used in the data generation for the Hard Transfer Dataset. Figure 4.5 also shows some representative plots for the types of trajectories in this dataset. Various transfers that navigate combinations of apse line rotations, transfers between circular and eccentric orbits, and orbital energy/SMA changes are represented in this trajectory data.



(a) Case 1

(b) Case 2

(c) Case 3

(d) Case 4

Figure 4.5: Sample Trajectories from Hard Transfer Training Data Set

Table 4.2: Hard Transfer Training Data Parameters

Variable	Type	Description	Value
TOF	Input	Time of Flight	1–1.9 years
I_{sp}	Input	Specific Impulse	3872 s
$T_{\max}$	Input	Max Thrust	1.33 N
μ_{sun}	Input	Central Body Gravitational Parameter	$1.3 \times 10^{11}\,\mathrm{km}^3/\mathrm{s}^2$
m_0	Input	Initial Spacecraft Mass	3366 kg
a_{initial}	Input	Initial orbit SMA	0.39–1.52 AU
e_{initial}	Input	Initial orbit eccentricity	0.0001–0.75
ω_{initial}	Input	Initial orbit argument of periapsis	0-360 deg
ν_{initial}	Input	Initial orbit true anomaly	0–360 deg
a_{final}	Input	Final orbit SMA	0.39–1.52 AU
e_{final}	Input	Final orbit eccentricity	0.0001–0.75
ω_{final}	Input	Final orbit argument of periapsis	0-360 deg
ϵ_i	Internal	Initial homotopic smoothing parameter	1
ϵ_f	Internal	Final smoothing tolerance	10^{-4}
γ	Internal	Smoothing rate	0.5
N_{traj}	Output	Number of trajectories generated	10,000
N_{vectors}	Output	Number of vectors per file	100

4.2 Two Body Transfer MDP

The proposed Two-Body Transfer (TBT) MDP is parameterized by a state that contains the spacecraft position, mass, and target states. This state should contain all the information the agent needs to achieve the target orbit successfully. Explicitly, the cartesian state vector used in the genera-

tion of training data is first converted to polar form:

$$\begin{aligned} r &= \sqrt{x^2 + y^2}, \\ \theta &= \text{atan2}(y, x), \\ v_r &= \frac{x v_x + y v_y}{r}, \\ v_\theta &= \frac{x v_y - y v_x}{r}. \end{aligned} \tag{4.1}$$

where θ is the argument of latitude of the spacecraft's current orbit and is defined as the angle between a spacecraft's position vector and a reference frame. The target orbit is defined by a target semi-major axis, eccentricity, and argument of periapsis (a_t, e_t, ω_t) respectively. Rather than including these Keplerian orbit parameters in the state space, the target orbit is instead represented in the state as the radius, radial velocity and tangential velocity components that the spacecraft would have if it was on the target orbit at its current argument of latitude. Figure 4.6 is a visualization of this target state vector representation. In this example, the spacecraft is currently in a near-circular orbit and the target orbit is a larger, more eccentric trajectory than the spacecraft's and is rotated by the argument of periapsis. In this re-framing of the target state, the polar state components of the spacecraft are compared with the polar state components of a proxy state that exists at the same argument of latitude on the target orbit. The true anomaly, ν_t, can be computed via Equation 4.2. Once ν_t is known, the target radial state components of interest (r_t, $v_{r,t}$, and $v_{\theta,t}$) can be computed directly assuming two-body dynamics and a_t, e_t, and ω_t. Finally, the state vector includes the mass of the spacecraft, m, as well as a time-to-go parameter, t_{tg}, which provides the agent information on how much time remains in the transfer.

$$\nu_t = \theta - \omega_t \tag{4.2}$$

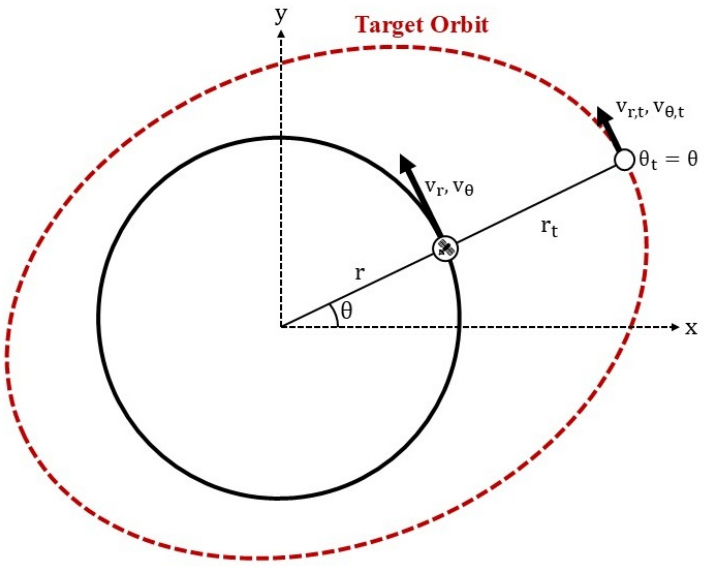


Figure 4.6: Transfer Target State Formulation

A collection of feature engineering steps are taken to minimize numerical instability for the neural networks during training. These steps include normalization by the following characteristic quantities:

$$
\begin{aligned}
l_* &= a_{\text{Earth}} \approx 1.496 \times 10^{11} m \\
m_* &= m_0 = 3,366 kg \\
t_* &= \sqrt{\frac{l_*^3}{G m_{sun}}} = 5,023,281 s \\
G &= 6.674 \times 10^{-11} \frac{m^3}{kg \times s^2}
\end{aligned}
\tag{4.3}
$$

where the characteristic length l_* is taken as the semi-major axis of Earth's orbit, m_* is the initial spacecraft mass at the beginning of the transfer. The characteristic time can be computed with the expression in Equation 4.3. With these characteristic quantities for length, mass, and time, the system

is normalized by applying the following transformations:

$$r' = \frac{r}{l_*}, \quad v' = v\frac{t_*}{l_*}, \quad m' = \frac{m}{m_*}, \quad t' = \frac{t}{t_*} \tag{4.4}$$

Furthermore, the sine and cosine of the argument of latitude are used as the input features instead of θ directly to minimize behavior discontinuity when the angle wraps between $0 \rightarrow 2\pi$. With this, we have a total of ten components in the state vector, with the final form of s_i listed in Equation 4.5.

$$\mathbf{s_i} = [r', \cos\theta, \sin\theta, v'_r, v'_\theta, m', r'_t, v'_{r,t}, v'_{\theta,t}, t'_{tg}] \tag{4.5}$$

The action space for the TBT MDP consists of the same three components from the optimal control problem, the throttle control u, and the two thrust direction vector components α_x and α_y. The thrust direction is rotated to polar components in the MDP for consistency, so the action vector $\mathbf{a_i}$ is:

$$\mathbf{a_i} = [u, \alpha_r, \alpha_\theta] \tag{4.6}$$

The reward function consists of three components: a position term, a velocity term, and a throttle term — each weighted by a unique, user-defined hyperparameter (w_p, w_v, and w_u, respectively). This particular function is not an authoritative choice for this problem, and future work should be dedicated to tuning reward parameters and features so as to maximize the desired behavior of the agent. The reward function computation is provided in Equation 4.7.

$$
\begin{aligned}
r_{\text{pos}} &= \exp\left(-w_p \Delta r'^2\right) - 1 \\
r_{\text{vel}} &= \exp\left(-w_v \Delta v'^2\right) - 1 \\
r_{\text{throttle}} &= -w_u u \\
r_{\text{tot}} &= r_{\text{pos}} + r_{\text{vel}} + r_{\text{throttle}}
\end{aligned}
\tag{4.7}
$$

where the $\Delta r'$ and $\Delta v'$ terms are the normalized position and velocity differences between the spacecraft and the target orbit as discussed earlier:

$$
\begin{aligned}
\Delta r' &= ||r'_t - r'|| \\
\Delta v' &= \sqrt{(v'_{r,t} - v'_r)^2 + (v'_{\theta,t} - v'_\theta)^2}
\end{aligned}
\tag{4.8}
$$

Intuitively, Equation 4.7 corresponds to penalizing the agent for being far from the target proxy state at each timestep. The agent is also penalized for using throttle, which is intended to reduce fuel waste to approach mass-optimal control. If the spacecraft is at the target orbit and not maneuvering, the reward will be at its maximum of zero.

The transition function for the MDP is governed by the equations of motion outlined in Equation 3.1. The spacecraft is propagated along its orbit subject to its thrust action until the next timestep is reached. The MDP episode terminates when the time of flight is exceeded, where the time of flight is chosen as the maximum of orbital periods of the initial or target orbit. Upon termination, the environment resets, which results in the spacecraft initializing into a new orbit and the target orbit changing such that the agent can learn from a diverse set of conditions and avoid overfitting. Table 4.3 contains the relevant parameters for the MDP resetting behavior, propagation, and reward computation.

Table 4.4 contains the list of hyperparameters assumed in this analysis for SAC training. These

Table 4.3: TBT MDP Parameters

Variable	Type	Description	Value/Range
step size	Env Parameter	Time between states/actions	2 weeks (fixed)
TOF	Reset Variable	Time of Flight	1-1.9 years
$a_{initial}$	Reset Variable	Initial Orbit SMA	1 - 1.52 AU
$e_{initial}$	Reset Variable	Initial Orbit Eccentricity	0.0001 (fixed)
$\omega_{initial}$	Reset Variable	Initial Orbit Argument of Periapsis	0 deg (fixed)
$\nu_{initial}$	Reset Variable	Initial Orbit True Anomaly	0-360 deg
a_{final}	Reset Variable	Final Orbit SMA	1 - 1.52 AU
e_{final}	Reset Variable	Final Orbit Eccentricity	0.0001 (fixed)
ω_{final}	Reset Variable	Final Orbit Argument of Periapsis	0 deg (fixed)
w_p	Reward Parameter	Scales Position Reward	0.2
w_v	Reward Parameter	Scales Velocity Reward	0.2
w_u	Reward Parameter	Scales Throttle Reward	0.02

parameters were kept as a close as possible to the set of parameters that Haarnoja et al. [32] list in their introductory SAC paper as they have been demonstrated to have success in a standard set of RL problems.

Table 4.4: RL Hyper-Parameters

Parameter	Value
Optimizer	Adam [50]
Learning Rate	0.0003
Discount Factor (γ)	0.99
Replay Buffer Capacity	10^6
Number of Hidden Layers (all networks)	2
Number of Neurons Per Layer	256
Training Batch Size	256
Activation Function	ReLU
Target Network Smoothing Coefficient (τ)	0.005
Target Network Update Interval	1
Gradient Updates Per Environment Step	1

Two key open-source python software libraries are being leveraged to streamline the for RL based training in this analysis and constructing the MDP environment. Gymnasium [51] is a standard RL problem interface that uses an expected format for packaging a problem as a Markov Decision Process (MDP) [52] in a software environment. A custom Gymnasium environment is used for the above TBT problem for integration in RL pipelines. Stable Baselines 3 is a library containing an implementation of various RL algorithms designed to be compatible with problems packaged as a Gymnasium environment [53] and it serves as the package responsible for implementing SAC in this analysis.

4.3 Two Body Rendezvous MDP

The TBR MDP follows a very similar structure to the TBT MDP, but has key differences to the state, action, reward and transition functions that allow for a rendezvous target. The first difference to note is that the state space was aggressively expanded to increase the likelihood of learning. Both cartesian and polar information is provided to the agent to reason over during the training process, with the expanded state becoming the following:

$$\mathbf{s}_i = \left[r_i',\ \cos\theta_i,\ \sin\theta_i,\ r_t',\ \cos\theta_t,\ \sin\theta_t,\ \mathbf{r}_i',\ \mathbf{v}_i',\ \mathbf{r}_t',\ \mathbf{v}_t',\ \boldsymbol{\Delta}\mathbf{r}',\ \boldsymbol{\Delta}\mathbf{v}',\ \|\boldsymbol{\Delta}\mathbf{r}'\|, \|\boldsymbol{\Delta}\mathbf{v}'\|,\ t_{\text{tg}}',\ m' \right] \quad (4.9)$$

The first six components describe the absolute radial position of both the target and chaser spacecraft. Absolute cartesian position and velocity vectors are also included for the target and chaser, along with the relative deltas in these quantities as well. The time remaining in the transfer, along with the mass of the spacecraft are the final two components in the state vector.

The action space remains completely unaltered from the TBT MDP implementation in Equation 4.6.

The reward function also follows the same structure as the TBT implementation in Equation 4.7, but instead of the transfer target orbit position and velocity residuals being used, the actual absolute magnitudes of the deltas in the position and velocity between target and chaser are used in the position and velocity components of the reward function.

The transition function uses the same equations of motion to propagate both the chaser and the target spacecraft states forward to the next step according to Equation 3.1. A majority of the

environment reset parameters are reused for the TBR problem. Table 4.5 shows the associated MDP parameters.

Table 4.5: TBR MDP Parameters

Variable	Type	Description	Value/Range
step size	Env Parameter	Time between states/actions	2 weeks (fixed)
TOF	Reset Variable	Time of Flight	1-1.9 years
$a_{initial}$	Reset Variable	Initial Orbit SMA	1 - 1.52 AU
$e_{initial}$	Reset Variable	Initial Orbit Eccentricity	0.0001 (fixed)
$\omega_{initial}$	Reset Variable	Initial Orbit Argument of Periapsis	0 deg (fixed)
$\nu_{initial}$	Reset Variable	Initial Orbit True Anomaly	0-360 deg
a_{final}	Reset Variable	Final Orbit SMA	1 - 1.52 AU
e_{final}	Reset Variable	Final Orbit Eccentricity	0.0001 (fixed)
ω_{final}	Reset Variable	Final Orbit Argument of Periapsis	0 deg (fixed)
$\nu_{initial}$	Reset Variable	Final Orbit True Anomaly	0-360 deg
w_p	Reward Parameter	Scales Position Reward	0.2
w_v	Reward Parameter	Scales Velocity Reward	0.2
w_u	Reward Parameter	Scales Throttle Reward	0.02

4.4 Converting Ephemeris Data to MDP Experience Tuples

After generating pre-optimal trajectory data and defining the necessary MDP structure that will be used in RL, the trajectories that will be used for supervised pretraining need to be converted from the ephemeris state vector format into a form that is compatible for RL. To achieve this, the trajectories were converted to the experience tuple form outlined in Section 4.2:

$$e_i = (\mathbf{s_i}, \mathbf{a_i}, r_i, \mathbf{s}_{i+1}, d_i) \tag{4.10}$$

For each state in the trajectory ephemeris, the equivalent MDP state is computed using the formulations in Sections 4.2 and 4.3, the optical control is supplied as the MDP action, and the resultant reward is returned. The next state in the mass-optimal ephemeris is taken as the transition state and the terminal flag d_i is true if the final state vector is the last state in the ephemeris, completing the necessary experience tuple. Figure 4.7 outlines this process for a sample trajectory. Once this conversion process

takes place, these experience tuples can be loaded into the SAC agent replay buffer.

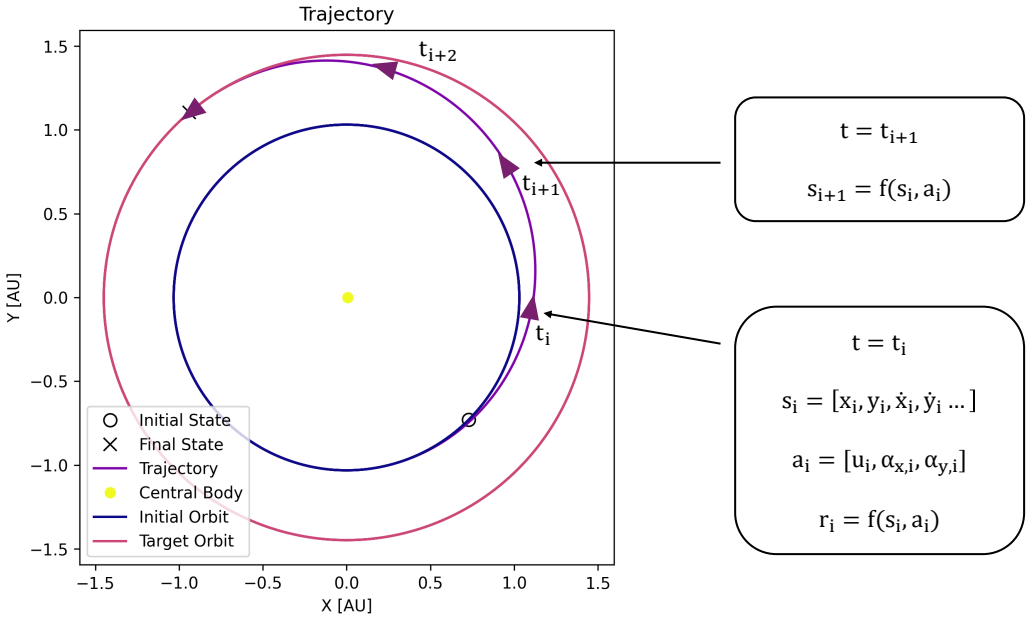


Figure 4.7: Generating Experience Tuples from Ephemeris Data

Once the experiences are loaded into the replay buffer, the SAC actor and critic networks sample from this replay buffer and perform gradient updates based on this data. This process is continued until the actor and critic losses stabilize, at which point we move to the standard SAC training process that uses environment steps to add experiences to the replay buffer while also continuing to perform gradient updates.

In this study, 1,000 mass-optimal trajectories are used, each containing 100 state-action pairs, resulting in a total of 100,000 experiences for the experience replay buffer. Gradient updates for the actor and critic networks were then performed on this replay buffer without an agent exploring the environment during pretraining.

4.5 Supervised Pretraining Stack Compatible with SAC

Each of the individual components have been defined for a generating mass optimal trajectory data and converting that data into MDP experience tuple form. The baseline SAC RL algorithm that will be used in the training experiments to follow has also been outlined in section 3.6. What remains is combining these components together into a workflow that successfully allows for the fusion of supervised learning and reinforcement learning. Summarizing this process at a high level, the first step is the parallel mass-optimal trajectory generation component. The second step is converting that ephemeris dataset into an experience replay buffer compatible with reinforcement learning. Step three involves conducting supervised learning using the SAC algorithm neural network structure, updating the actor, value, and critic network components with the loaded replay buffer data. Figure 4.8 illustrates these three major pretraining steps.

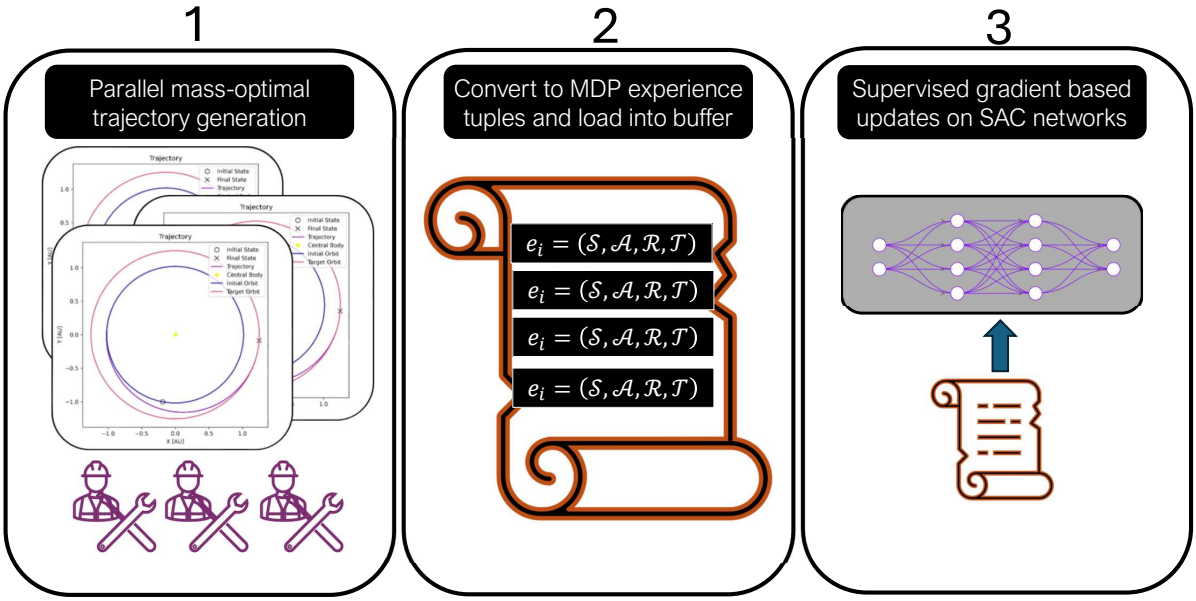


Figure 4.8: Supervised Training Stack Diagram

The supervised learning step is analogous to just conducting the second half of the SAC algo-

rithm outlined in Algorithm 1, in other words, the exploration step is temporarily removed in pre-training, and the process consists of just updating the neural networks. Once the networks have been sufficiently trained on the fuel-optimal replay buffer, the pretraining process is complete. The pre-trained model can then be used as a starting point for the standard SAC RL algorithm to determine the performance and training speed benefits of this process.

4.6 Monte Carlo Agent Evaluation Method

A Monte Carlo approach to agent evaluation will be used during the supervised pretraining process, the reinforcement learning process, and also after training has been completed to quantify the expected performance of the agent's learned policy. A trade-off exists between the number of evaluation trials to perform during training as well as the frequency of that evaluation process against the time spent actually training. Evaluation cycles are computationally expensive, the agent must navigate the environment across potentially many episodes. This time spent evaluating the model could be time spent actually training and improving the agent's performance. On the other hand, the SAC RL process is a stochastic one, and frequent agent checkpoints are beneficial for providing visibility in the learning process.

During the pretraining and reinforcement learning process, 100 episodes are used as the baseline model rollout extent per evaluation. An episode here denotes one trajectory, where the agent begins in an initial reset condition defined by the parameters in the MDP formulations and steps through the environment until a terminal condition is reached. The main statistic tracked during training is the accumulated final reward of the agent. The mean and standard deviation of these final rewards achieved during each episode are recorded. During training, if the agent reaches a new maximum

reward, the model is saved and this new max achieved performance becomes the new standard. Future agent models are only saved if this performance is improved upon in a subsequent model evaluation. At the end of training, the best achieving model (and not necessarily the model at the completion of training) will be passed on to the next step in the learning process.

The frequency of these evaluation runs was varied based on the training speed, and there was not a consistent evaluation interval enforced during the training process for greater flexibility during training. A lower bound of 25 model evaluations per training run was used as a general rule of thumb in this analysis.

After the pretraining and baseline RL processes were completed, a much more extensive Monte Carlo evaluation run was conducted on the resultant agent model to more fully analyze the expected performance. For this larger Monte Carlo, four major metrics are tracked during the model evaluation process: final accumulated reward, the final mass fraction of the spacecraft, and the terminal position and velocity residuals with respect to the target orbit. For this comprehensive Monte Carlo analysis, 10,000 trials are used to fully capture the expected performance of the agents.

Chapter 5: Pretraining Experiment for the Circular TBT Problem

An initial experiment using the Circular Transfer Dataset from Subsection 4.1.1 was intended to serve as a proof of concept for the supervised pretraining stack. For this experiment, an agent that has been pretrained on the Circular Transfer Dataset and an agent with no pretraining underwent SAC reinforcement learning. The pretrained agent was checkpointed at several points during the pretraining process and used in baseline RL runs later on, so while multiple curves are shown in the baseline RL plots, it is really the same agent with increasing levels of pretraining. An assessment of the final performance of these agents during and after SAC RL was conducted to measure the effects of pretraining using the Monte Carlo approach explained in Section 4.6. The reader may refer back to the SAC hyper-parameters assumed in this training process in Table 4.4 for the RL configuration assumed for this set of SAC runs.

In this first experiment, the agent performance was evaluated using a rollout of 100 episodes every 10,000 training steps during the supervised pretraining and traditional reinforcement learning processes. This evaluation reward, along with the standard deviation from each of these rollouts during this pretraining process, is shown in Figure 5.1. Each red dot on this evaluation reward plot signifies a new maximum reward achieved when performing the evaluation rollout during pretraining. At each of these points, a model checkpoint is saved for further analysis and training.

Figure 5.1 shows the mean reward quickly increasing from a value of approximately -12 to -4 within the first 50,000 pretraining iterations. After this, there are diminishing returns in the reward improvement, with no substantial improvement in the total reward after 500,000 pretraining iterations. The extent of pretraining is an area of future study, but for this experiment we assume one million pretraining steps (gradient updates) prior to reinforcement learning due to clear stabilization of the

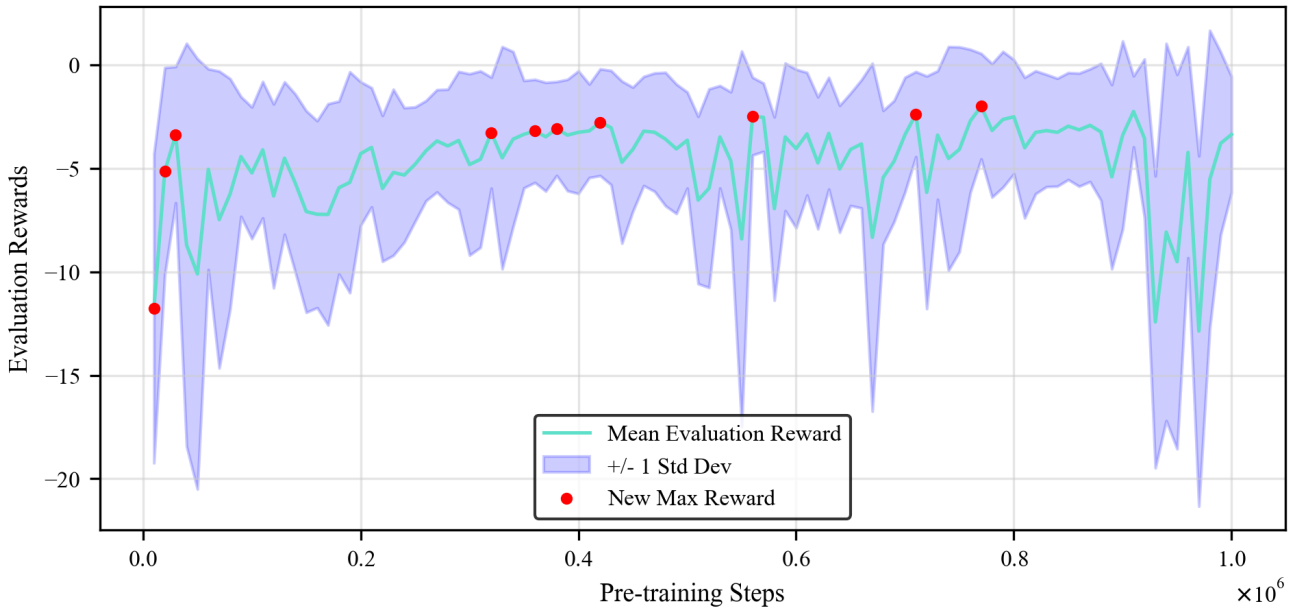


Figure 5.1: SAC Model Reward During pretraining

evaluation rewards and actor and critic losses (refer to Figure 5.2). This stability implies the networks have converged to a locally optimal policy for the pre-generated trajectories, and can move on to the RL pipeline with the updated controller.

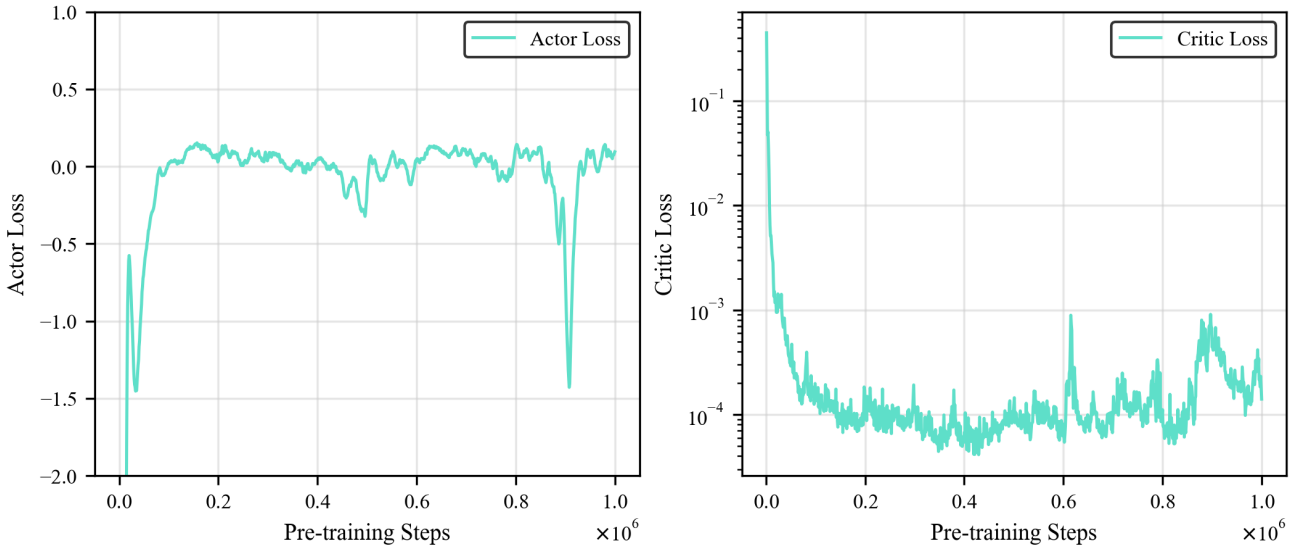


Figure 5.2: SAC Pretraining Actor/Critic Losses

After pretraining the agent, five agents are trained in the standard SAC pipeline to compare

the performance and training efficiency benefits of supervised pretraining. The first agent is trained from scratch, with no pre-populated replay buffer or pretrained model to begin the training process. The remaining four agents are initialized with the model from selected checkpoints in the pretraining process. A total of five million training gradient updates are performed for each of these agents, with each update corresponding to one step in the training environment. The average reward per episode during training is shown in Figure 5.3 for the default and pretrained agents. The most pretrained agents are able to quickly synthesize a control profile that reaches a maximum reward limit eight times faster than with standard RL alone — i.e. the agent pretrained with 760k iterations reaches a mean average reward per episode of negative one at approximately 100,000 iterations while traditional SAC takes over 800,000 iterations to reach this point.

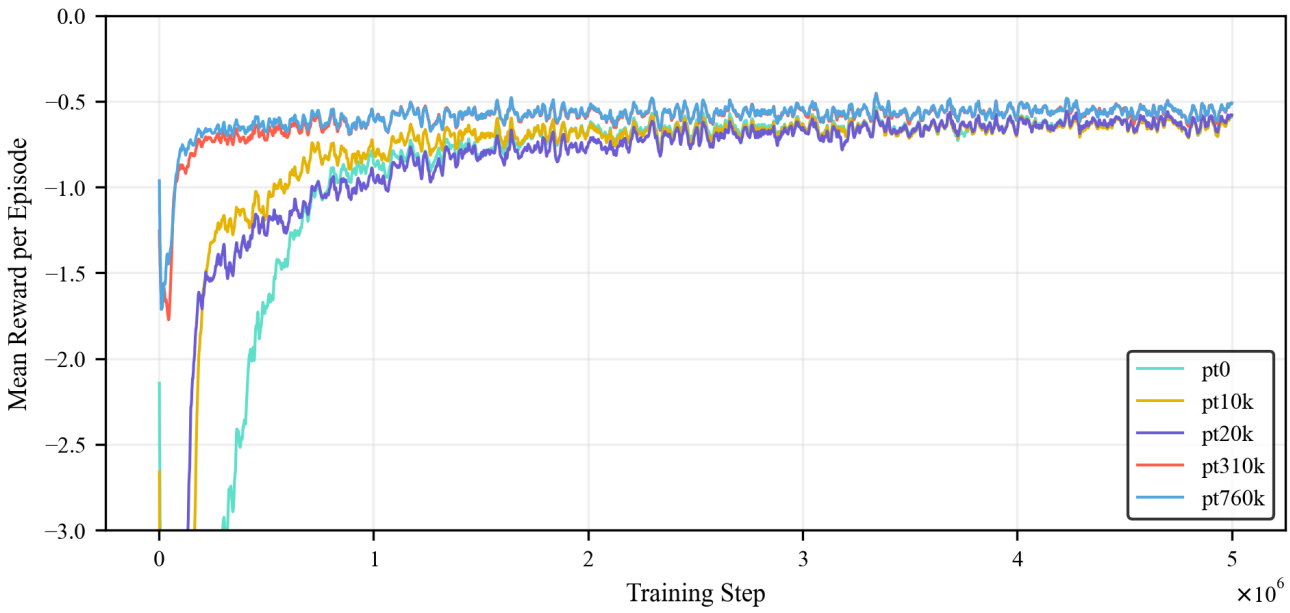


Figure 5.3: SAC Reward During Training

There are substantial differences in the actor and critic network losses from the different approaches made evident in Figure 5.4. In these plots, the pretraining gradient updates are counted toward the total steps on the x-axes for the pretrained agent. The actor loss represents the belief in how

good the current policy is at selecting actions that maximize value according to the critic networks. This loss needs to be considered in tandem with the critic loss, which measures the amount of error the critic networks experience when estimating the values of states. A steady decrease in the critic loss is evident in both the pretrained and vanilla RL runs, with the pretrained getting a head start of approximately two orders of magnitude lower (better) than the vanilla RL agent. Training is halted when the actor and critic network losses stabilize.

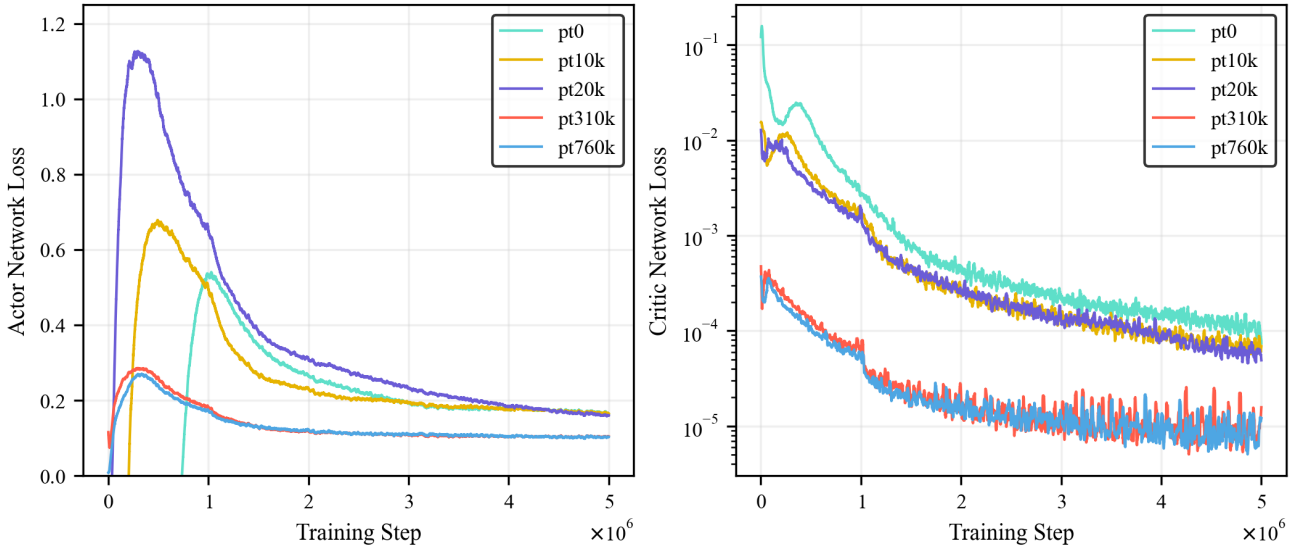
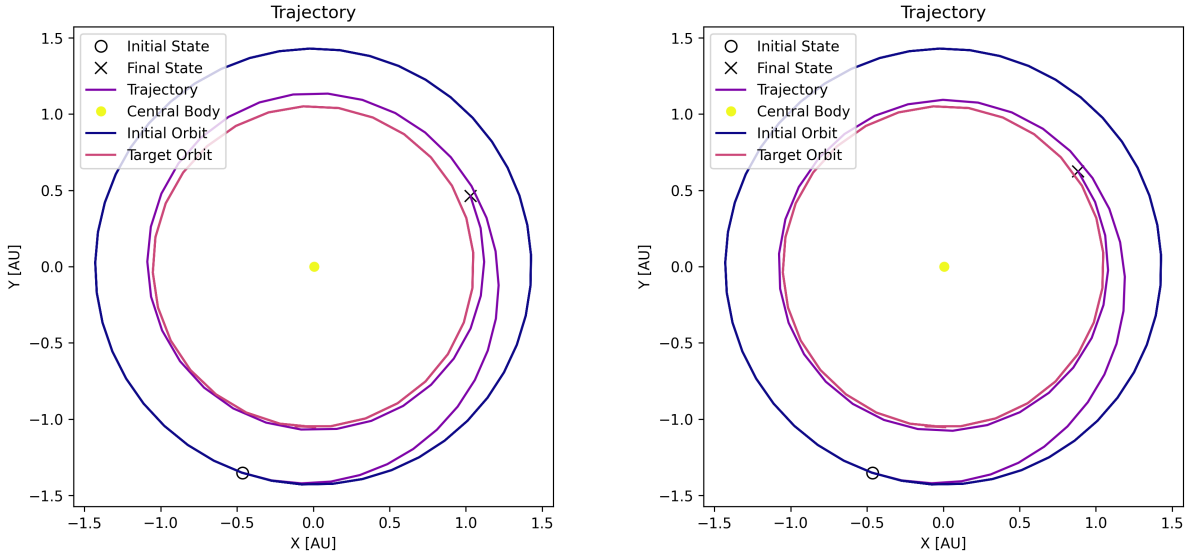


Figure 5.4: SAC Actor Loss During Training

After SAC training, the resultant controllers are validated using 10,000 Monte Carlo rollouts. Sample trajectories from the rollouts produced by the traditional and pretrained controllers are shown in Figure 5.5. The agent performance is assessed using average reward per episode, the final mass fraction of the spacecraft at the conclusion of the time of flight, and the final position and velocity residuals with respect to the target orbit. The mass, position residual, and velocity residuals are non-dimensionalized with the same characteristic units in Equation 4.3. Figure 5.6 displays histograms for the performance of the two agents.

In the reward plot histogram, the pretrained agent is shown to outperform the standard RL agent,



(a) Vanilla RL

(b) Pretrained (760k iterations)

Figure 5.5: Sample Trajectories using Vanilla RL and Pretraining

as the rewards are more tightly grouped and closer to zero, the theoretical maximum reward possible. The mean reward for the supervised agent is -0.565 whereas the mean rollout reward for the traditionally trained agent is -0.621. There is less of a clear indication with the mass fraction distributions, but analysis of the trajectories suggests that pretrained agents are willing so spend slightly more fuel to maximize reward and achieving the target orbit earlier compared to agents with no pretraining. The terminal position and velocity residuals are the differences between the final orbit achieved by the agent and the target orbit. The pretrained model has a tighter distribution of position and velocity residuals, and these are closer to zero as well. The mean reward, final mass fraction, and terminal position and velocity values are provided in the legend of each histogram.

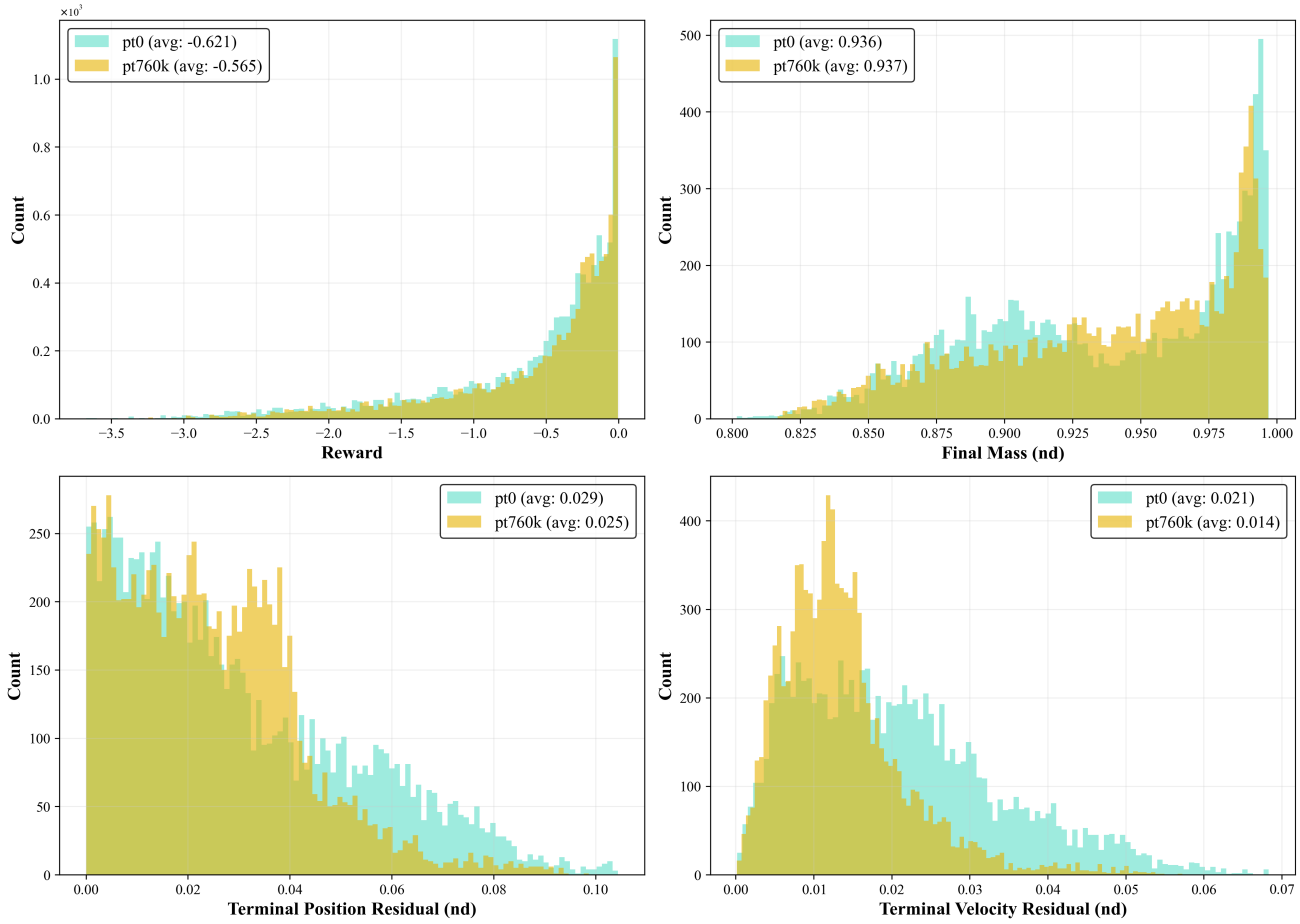


Figure 5.6: Monte Carlo RL Controller Evaluation (10,000 rollouts)

Chapter 6: Pretraining Experiment for the Hard TBT Problem

The second major experiment involves an additional aspect of complexity to the classes of orbits considered. The previous experiment was limited to just circular orbits between the SMA ranges of Earth and Mars, the initial and target boundary conditions for this second experiment now include variation in eccentricity between 0 and 0.75 and in argument of periapsis in the full range of 0 to 360 degrees. This means that agents in this experiment will need to reason over apse-line rotations, and eccentric orbital shaping to achieve the targets. The added eccentricity also presents more of a risk of poor agent behavior leading to hyperbolic trajectories by randomly performing large prograde maneuvers near apoapsis.

For this study, 10,000 transfer trajectories were generated for pretraining. A two-parameter sensitivity study is conducted here, with one independent axis being the amount of reference training data available in the replay buffer, and the other axis being the duration of pretraining. Two pretraining runs were conducted with one thousand and ten thousand reference trajectories loaded into the replay buffer. Each pretraining run was conducted for a total of half a million gradient updates. Every twenty thousand steps, a MDP reward evaluation process takes place. The agent, as in the previous pretraining experiment with the TBT Circular problem, is evaluated while traversing one hundred low-thrust trajectories. If the model achieves a new mean maximum reward during the evaluation process, an agent checkpoint is saved off and training continues. After pretraining takes place, the resultant agent is passed to the standard SAC process with 1 million training iterations for each case. The number of SAC training cycles is reduced here (the previous experiment was conducted with 5 million training iterations) due to the increased number of cases being considered for this study. Figure 6.1 shows the evaluation rewards during the pretraining process for these three replay buffer scenarios.

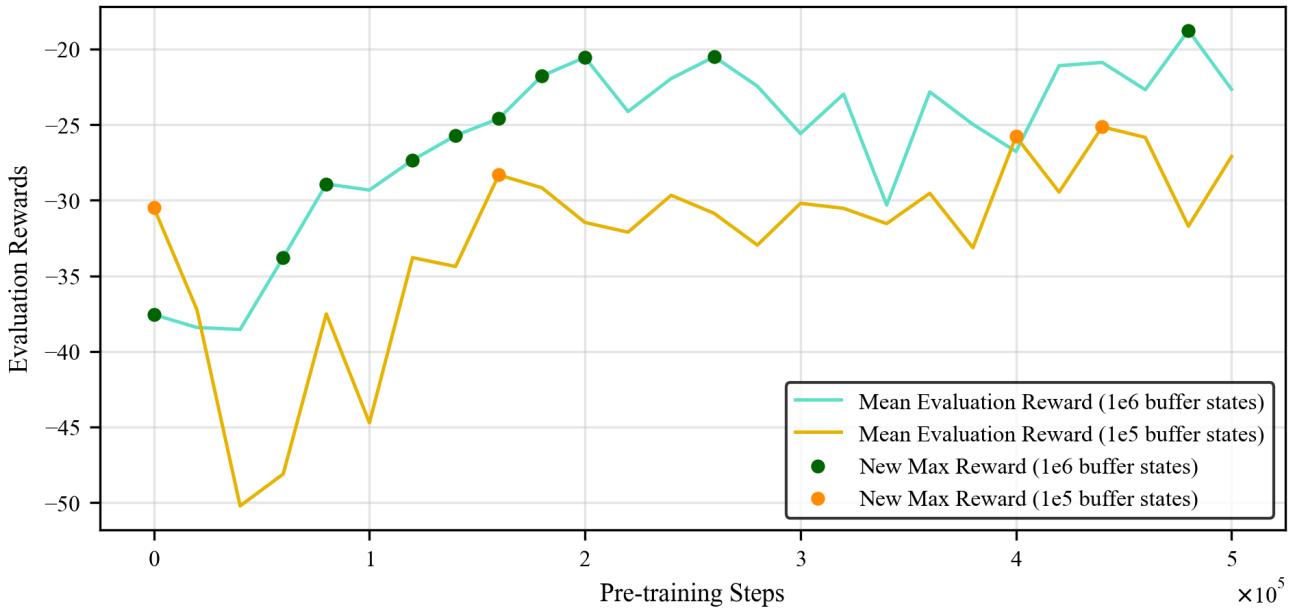


Figure 6.1: SAC Model Reward During pretraining

Each new maximum reward is highlighted by a marker in the figure, corresponding to an agent checkpoint. Pretraining limits are also chosen here to evaluate the extent of pretraining for each of the buffer size initialization scenarios. Three limits are chosen based on the distribution of agent checkpoints: 0.1 million, 0.25 million, and 0.5 million pretraining iterations. Whatever the most recent best performing agent is at each of these checkpoints is saved for further analysis.

The agent actor and critic losses for each of these buffer size scenarios is provided in Figure 6.2. The actor loss plot shows a lower loss for the smallest buffer size pretraining run, with higher loss curves shown for the one million buffer size run. The critic loss plot shows very interesting behavior. The 1e5 replay buffer run experiences the smaller and continually decreasing critic loss, despite the lower evaluation rewards as shown in Figure 6.1. This suggests the agent is learning all it can for its value estimates due to the smaller amount of training data and is overfitting to an insufficient amount of experiences. The gradual net increase in the 1e6 critic loss after the initial drop suggests there is potentially more training to be done.

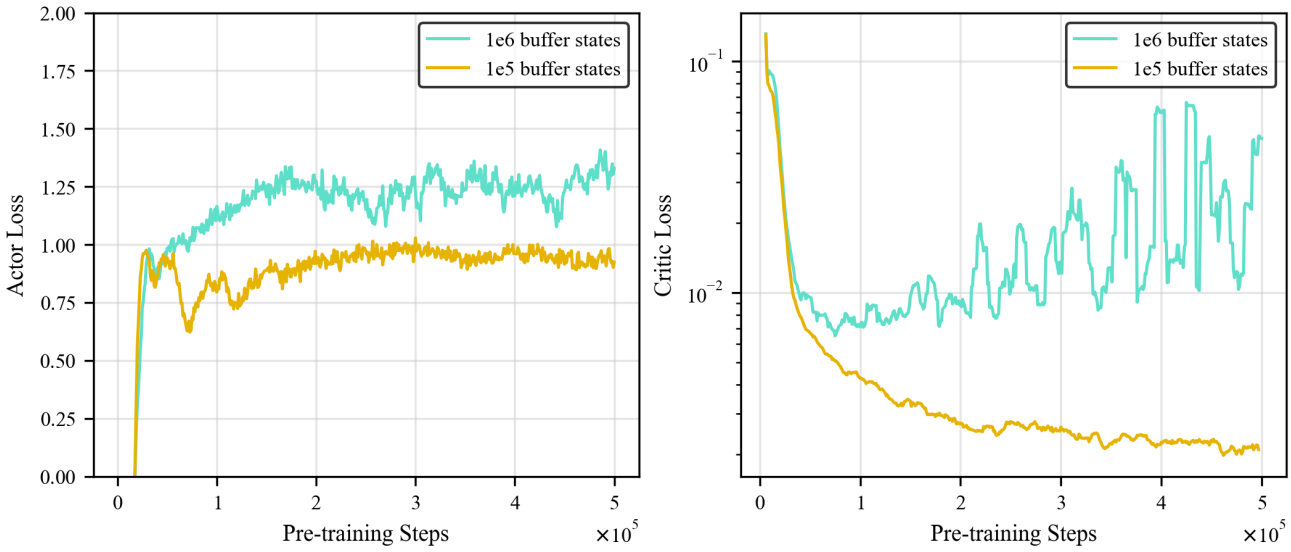


Figure 6.2: SAC Pretraining Actor/Critic Losses

There are a total of six combined configurations when considering the three pretraining limits (0.1 million, 0.25 million, and 0.5 million pretraining iterations) for both of the replay buffer scenarios (0.1 million, and one million experience tuples in the replay buffer). The average reward per step during SAC training is provided in Figure 6.3. The agent pretrained with the least amount of data available to the agent, along with the least amount of time spent training, is the outlier in this analysis. The other runs are fairly within family with respect to the episode reward during training. This suggests that there may be some minimum amount of training data and training iterations required to really accelerate the RL training process. For this experiment, the agent reaches a point of diminishing returns somewhere between 100k iterations and 250k iterations for the smaller replay buffer containing 1,000 trajectories. All three agents have similar reward profiles for the larger replay buffer, indicating that enough data in the replay buffer can balance out fewer training iterations.

A plot of the actor/critic losses during the standard SAC training (not be confused with pre-training) for the these six cases is shown in Figure 6.4. This plot separates the two replay buffer sets of runs, with the smaller 1e5 replay buffer actor and critic losses plotted in the top row of the figure

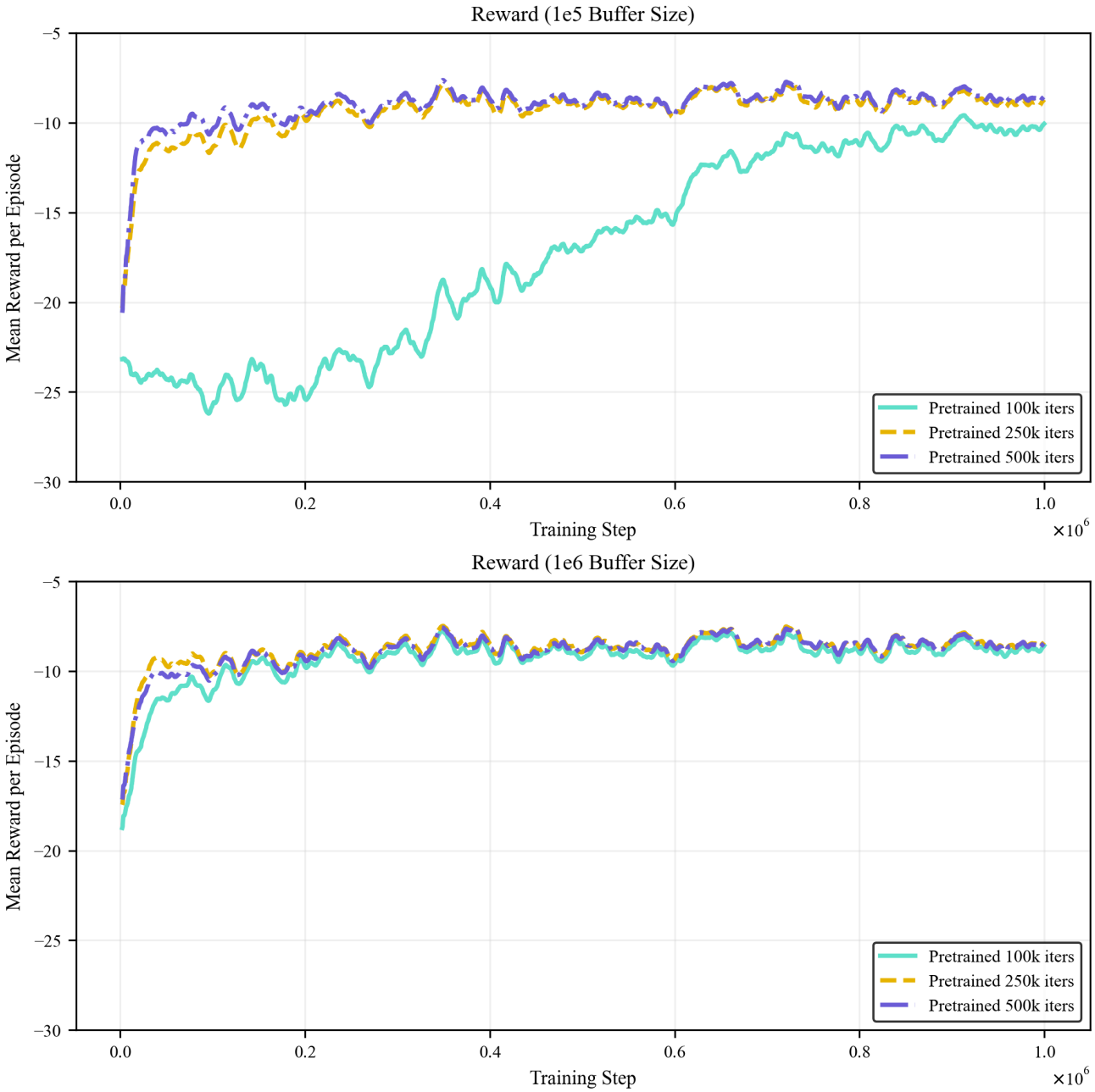


Figure 6.3: SAC Reward During RL Training for the Small Buffer Pretrained (Top) and the Large Buffer Pretrained (Bottom) TBT Hard Cases

and the larger 1e6 replay buffer actor and critic losses plotted on the bottom. In the standard training segment shown, the actor losses for the smaller and larger buffer cases follow the same trend and are generally closely packed, with the same exception of the scenario with the least amount of training and the smallest buffer size (the 1e5 buffer pretrained with 100k iterations). This seems to indicate

that pretraining is valuable for this problem, but only up to a point, with 250k iterations achieving comparable actor/critic losses with the 500k cases across the two buffer size runs. Modest reductions in the actor/critic losses are visible with the smaller buffer run, with the actor loss actually being higher and the critic loss essentially overlapping between the 250k and 500k runs for the larger buffer size.

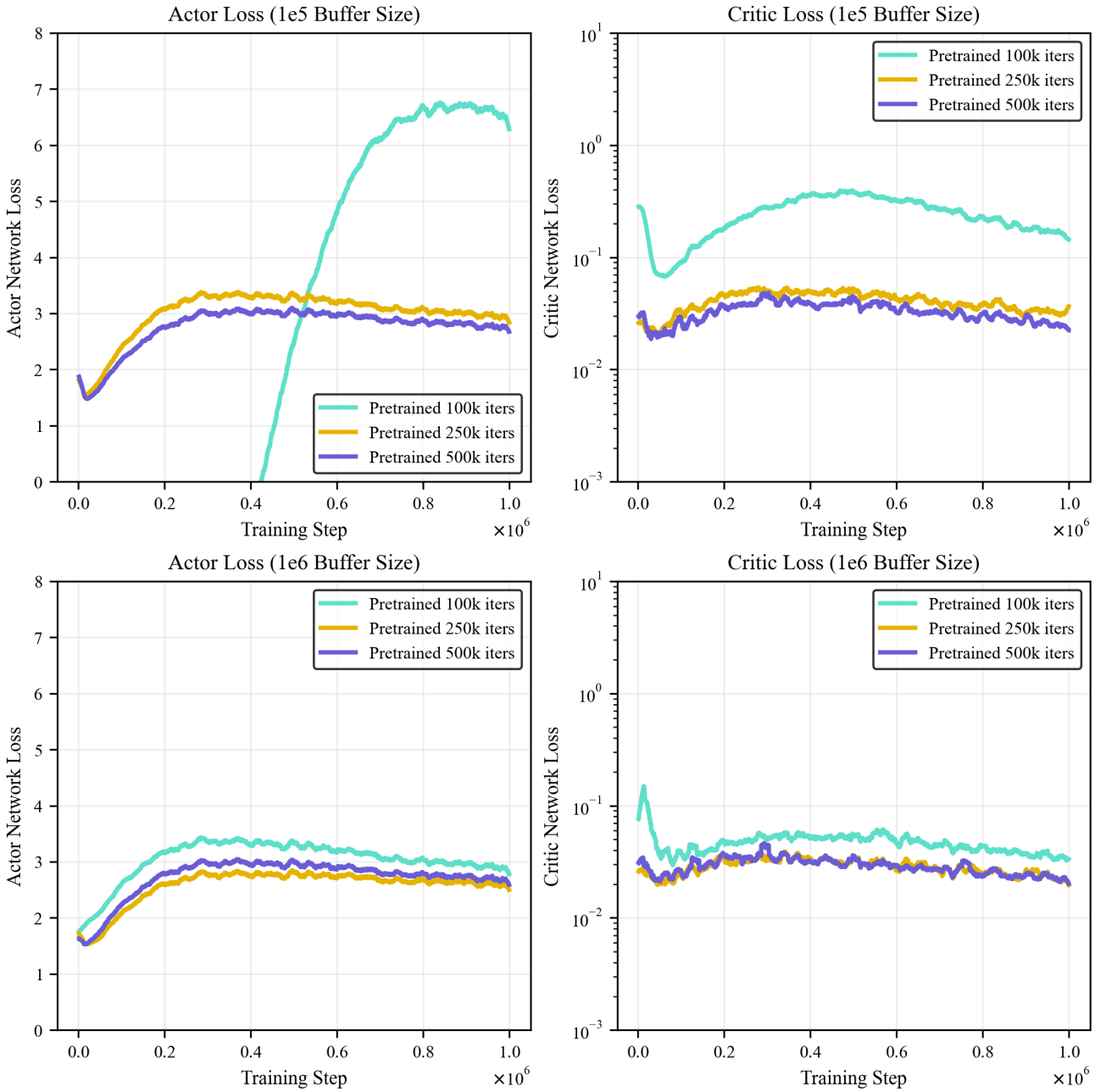


Figure 6.4: SAC Actor Loss During Training for TBT Hard Problem

A summary table of all of the Monte Carlo rollouts is provided in Figure 6.5. The same rollout evaluation process is used in each of the six trained agents, with the agent navigating 10,000 trajectories and the control responses and resultant rewards/statistics computed. Each entry in the table contains the same four evaluation metrics used in the TBT Circular problem: MDP reward, terminal mass fraction, and terminal position and velocity residuals with respect to the target orbit. The means for the trajectory reward, terminal mass fraction, final position residual, and final velocity residual are denoted by squares, circles, triangles, and diamonds in the figure respectively. These evaluation means are also color coded by performance. The complexities of training are highlighted by the mixed performance of the agent near the upper limits of pretraining. Overall, more training and more states in the replay buffer improves the performance across the board, but when two 500k pretrained agents are compared across the two replay buffer sizes, the smaller buffer agent outperforms the larger buffer agent drastically in terms of the position residual while maintaining a lower reward. This indicates the agent is maximizes its reward through some unintended behavior by saving on fuel consumption at the expense of final position residual with respect to the target.

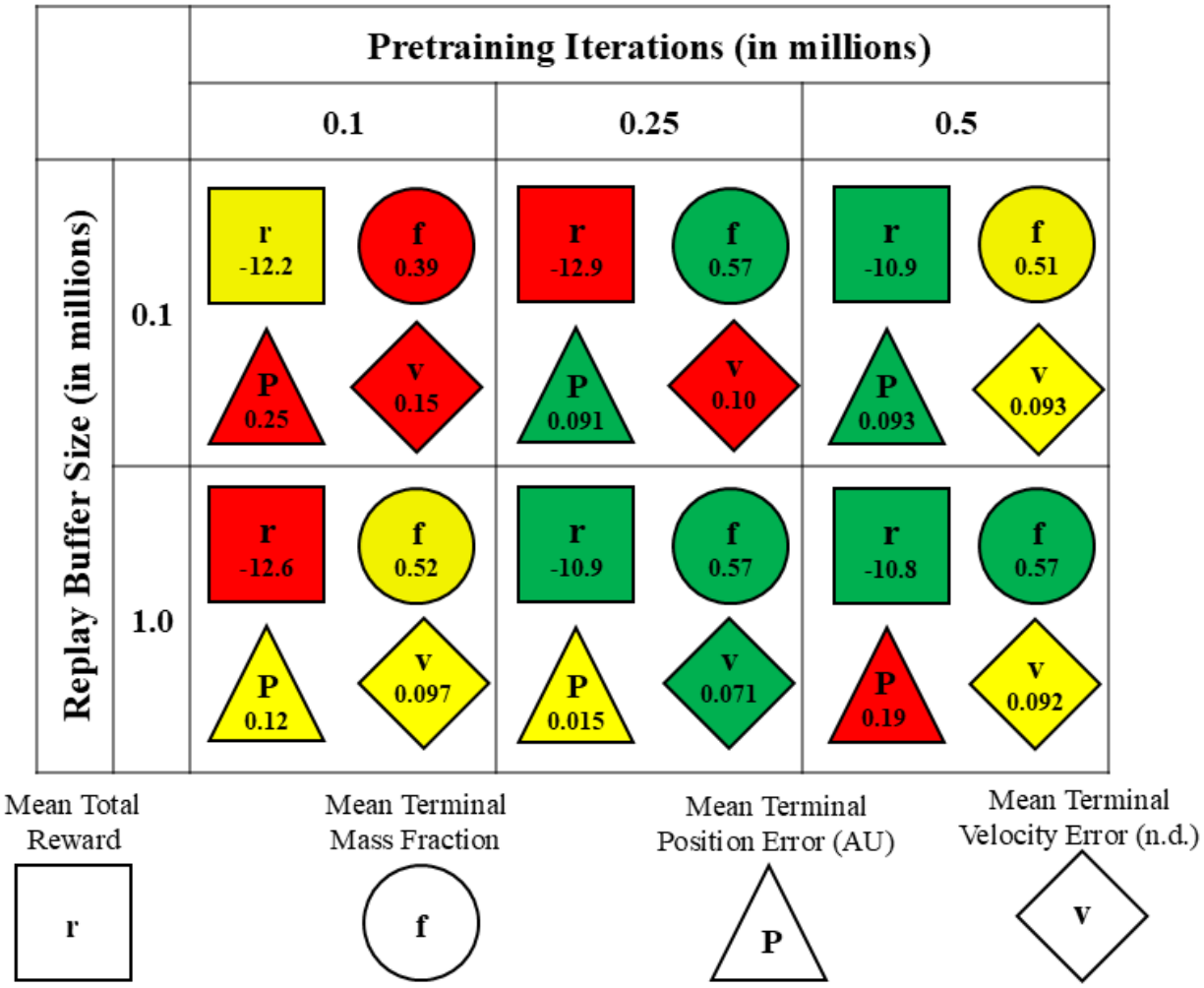
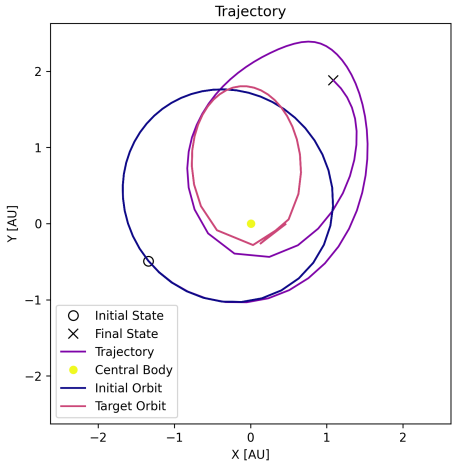


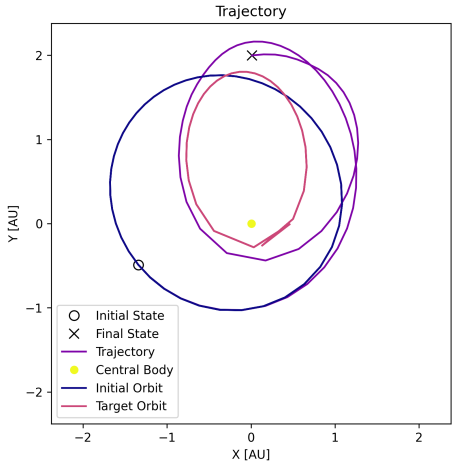
Figure 6.5: Monte Carlo Summary of TBT Hard Sensitivity Study

Individual rollouts with the six trained controllers are provided for the same initial environment seed in Figure 6.6. The trajectory sub–figures are organized from left to right by buffer size and top to bottom by pretraining extent. Case (a) is unsurprisingly the worst, having the least amount of pretraining time and data available to the agent during pretraining. Case (b) is able to improve slightly, navigating closer to the target orbit. The intermediate cases (c) and (d) improve on this by matching the shape of the target orbit earlier on in the transfer, and actually overlapping in position prior to the

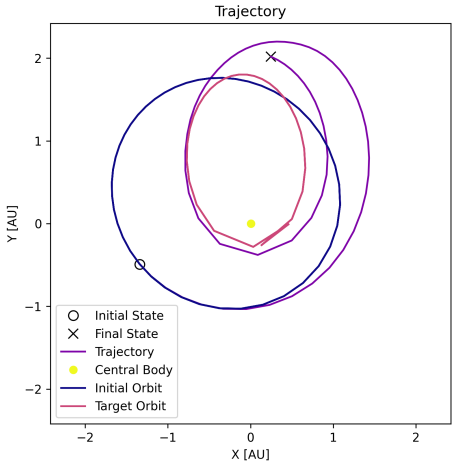
terminal state, but still have a substantial final position residual. Case (e) is the agent with the max pretraining iterations using the small buffer, and this agent is able to match the target Keplerian orbit very closely at the end of the transfer. Case (f) is the agent with the max pretraining and larger buffer size. This trajectory exhibits some peculiar behavior, and potentially evidence of reward function hacking. The agent accumulates reward earlier on as it closes in on the target orbit earlier. The wider swing to the upper right from the other trajectories is reduced here, meaning the agent gets closer to the transfer orbit earlier, minimizing the negative reward accumulated when the agent is far from the target. Additionally, this sweep to the upper right occurs near apoapsis, meaning the spacecraft is out in this region for a longer period of time. In the TBT MDP, reward is accumulated on a per-timestep basis, so longer coasting times away from the target are punished more. This is coming at the expense of the terminal position and velocity residual, but the agent is trained to maximize the expected sum of rewards, and has found a behavior in this instance that squeezes slightly more reward performance out of this particular trajectory. Future experimentation should account for this tendency for this problem.



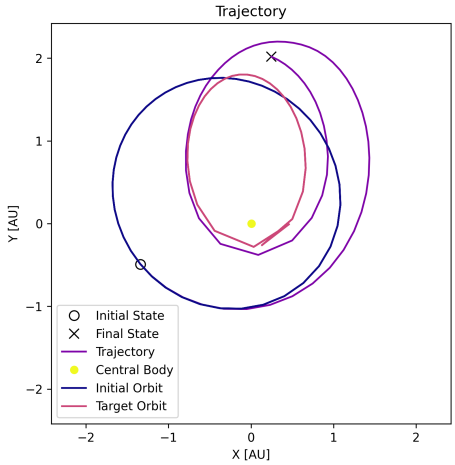
(a) 1e5 Exp in Buffer/1e5 Pretraining Iters



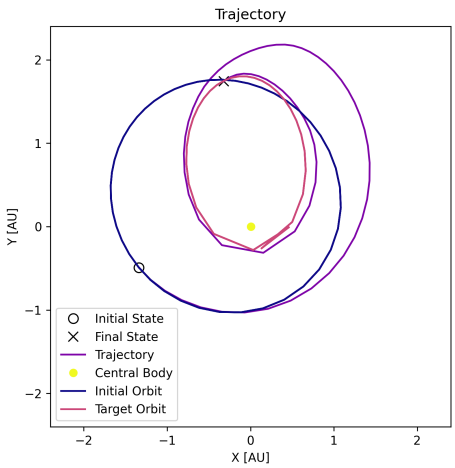
(b) 1e6 Exp in Buffer/1e5 Pretraining Iters



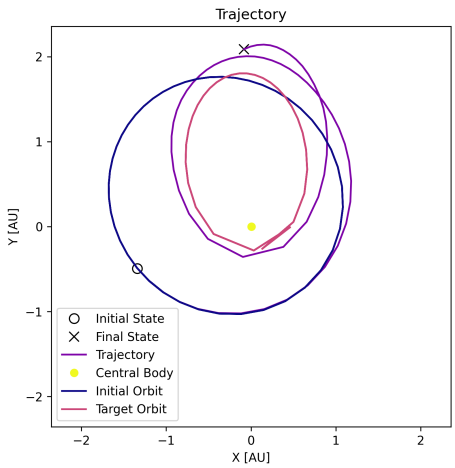
(c) 1e5 Exp in Buffer/2.5e5 Pretraining Iters



(d) 1e6 Exp in Buffer/2.5e5 Pretraining Iters



(e) 1e5 Exp in Buffer/5e5 Pretraining Iters



(f) 1e6 Exp in Buffer/5e5 Pretraining Iters

Figure 6.6: Sample Trajectories from the Hard Two-Body Rendezvous Training Data Set

Chapter 7: Discussion and Future Work

Seeding the replay buffer with mass-optimal trajectories leads to noticeable improvements in controller performance and training speed over RL in isolation. With supervised pretraining, the agent is provided a reference set of trajectories that persist in the replay buffer during pretraining because the replay buffer capacity is set to one million experiences. If the number of experience steps exceeds this capacity, the SAC implementation used in this work employs a first-in, first-out cycling behavior to replace older experiences. Replay buffer size is typically constrained by hardware limitations, and setting the buffer to excessively large sizes can retain stale data that may be suboptimal. The replay buffer capacity of one million experiences used by Haarnoja *et al.* [32] is adopted here as a relatively safe choice; however, further investigation of the effect of replay buffer size is warranted.

During supervised pretraining, the actor and critic networks are updated only using existing experiences in the replay buffer; however, a key component of a complete RL approach is interaction with (and exploration of) the environment. Each agent is given 50,000 time steps of environment interaction at the start of SAC training before gradient updates are applied to the actor and critic networks. For the baseline agent, this corresponds to executing random thrusting actions while populating the replay buffer. This ensures the agent begins training with a breadth of experience before applying gradient updates that could otherwise drive learning toward a poor local optimum.

The step size used in this analysis is set to a relatively large time span of two weeks. This choice caps the number of steps per episode to approximately 50, which is manageable for initial RL runs. In this setting, the training process benefits more from exposure to a diverse set of conditions across many episodes than from fewer episodes with a more granular step size. The large step size may affect the final terminal residuals; further studies should assess the trade-offs associated with varying step

size in this problem. With the initial baseline experiments completed, the step size can be reduced to allow for finer resolution in the RL state and action spaces.

Several avenues for future work follow from this analysis. First, a complete pretraining-stack implementation for the TBR problem is still in progress. An extensive set of fuel-optimal rendezvous trajectories has been generated for use in pretraining. Examples of these trajectories are shown in Figures 7.1 and 7.2. Figure 7.1 contains rendezvous trajectories for circular initial and final orbits, similar to the circular transfer dataset, but with the added constraint of matching true anomaly at the end of the designated time of flight. Figure 7.2 shows rendezvous trajectories with additional variation in eccentricity and argument of periapsis, increasing difficulty in a manner analogous to the hard transfer dataset.

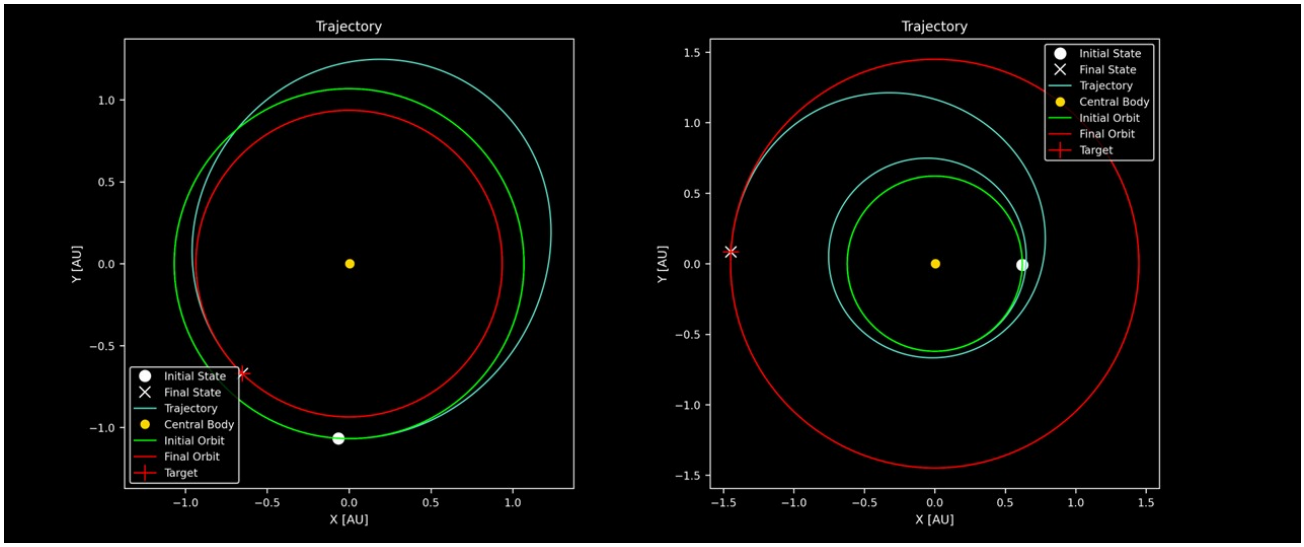


Figure 7.1: Circular Rendezvous Training Data Sample Trajectories

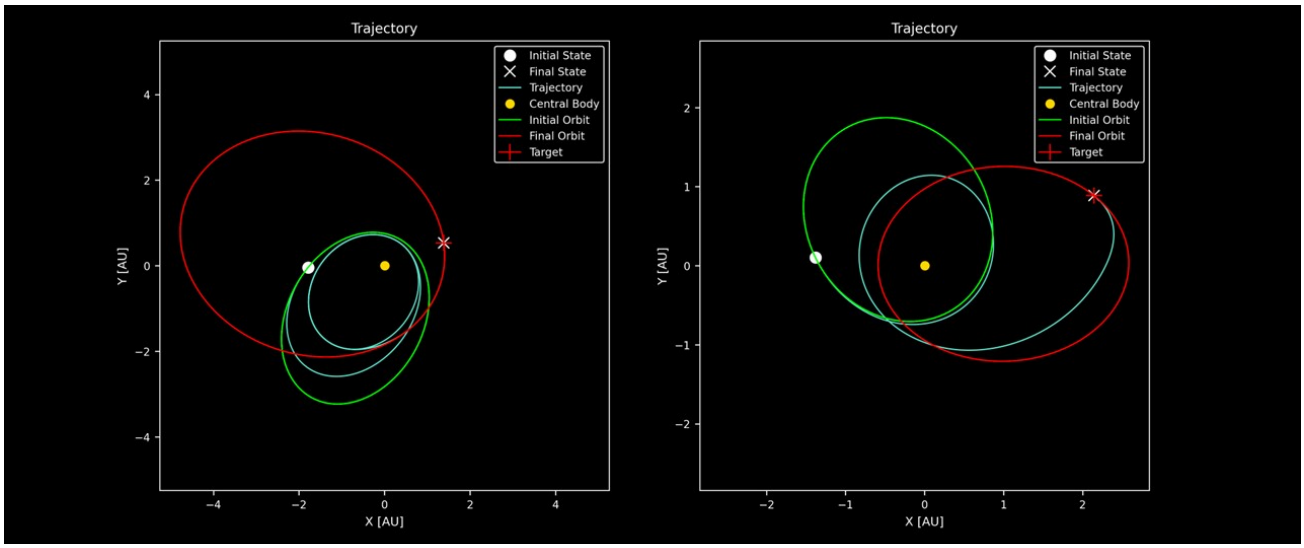


Figure 7.2: Hard Rendezvous Training Data Sample Trajectories

Baseline SAC training runs have been conducted successfully using the TBR MDP formulation in Section 4.3. A sample reward history from a 10-million-step SAC training run is provided in Figure 7.3. The next step is to complete the pretraining-stack implementation for TBR and quantify the utility of pretraining on this more challenging class of trajectory problem.

Figure 7.3: Hard Rendezvous Training Data Sample Trajectories

Increasing the complexity of the TBT and TBR problem classes is also of interest for future

work. More realistic dynamics could be incorporated to improve mission applicability, such as third-body or full N-body effects. Extending the formulation to three dimensions in both the state and action spaces is also warranted.

Refinements to the pretraining process are likewise needed. The TBR problem appears to require substantially more pretraining time and/or data to achieve performance gains comparable to those observed for TBT. Reward shaping for the TBT and TBR MDPs may be required to better align learned behavior with realistic mission goals. For example, a time-dependent term could be introduced in the reward function to increase the weight of terminal accuracy as the spacecraft approaches the end of the time of flight. This would be expected to encourage smaller terminal position and velocity residuals for both problems. Preliminary efforts have also begun on automation improvements to accelerate experimentation and training, which will become increasingly important as problem complexity and computational demands grow.

Chapter 8: Conclusion

This thesis investigated a hybrid learning pipeline for low-thrust trajectory design that combines supervised pretraining on fuel-optimal reference trajectories with subsequent reinforcement learning using the Soft Actor-Critic (SAC) RL algorithm. The central motivation was to retain the physics-consistent structure and accuracy of optimal control solutions while enabling fast, closed-loop guidance policies that can be evaluated efficiently and can adapt through continued interaction. To isolate the effect of pretraining and quantify its impact, the study focused primarily on the Two-Body Transfer (TBT) problem under simplified two-body dynamics, and planar motion with perfect state knowledge.

High-fidelity mass-optimal reference trajectories were generated using an indirect optimization method with homotopic smoothing to robustly handle a bang–bang control profile. These trajectories were then converted into experience tuples consistent with the proposed Markov Decision Process (MDP) formulations and used to initialize and pretrain the SAC actor and critic networks through replay-buffer sampling prior to online RL. In this framing, supervised pretraining serves as an informed initialization that reduces the exploration burden of long-horizon continuous-control learning while preserving SAC's stochasticity and ability to improve through reinforcement learning.

Experimental results for the circular TBT problem demonstrate that supervised pretraining can substantially reduce SAC sample complexity. The pretrained agent reached comparable reward levels roughly an order of magnitude faster than training from scratch, and Monte Carlo evaluation over 10,000 rollouts showed improved mean reward and tighter distributions in terminal position and velocity residuals relative to the vanilla RL baseline. For the harder TBT problem with increased eccentricity and argument-of-periapsis variation, a sensitivity study indicated that both the amount of training data available in the replay buffer and the duration of pretraining materially affect downstream RL perfor-

mance. In particular, moderate pretraining with sufficient demonstration coverage generally improved training stability and rollout performance, while some configurations exhibited behavior consistent with reward exploitation, emphasizing that reward design and evaluation metrics must be carefully aligned with mission objectives.

Overall, these results support the central premise of this thesis: that seeding reinforcement learning with supervised pretraining on optimal-control demonstrations and can yield faster learning and improved terminal accuracy in low-thrust trajectory design tasks, while still leveraging reinforcement learning's strengths. At the same time, the observed sensitivity to reward shaping, replay-buffer composition, and pretraining extent highlights that hybrid pipelines require careful design choices to avoid overfitting to demonstrations or converging to policies that optimize surrogate rewards without improving hollistic performance.

Future work should extend this framework toward higher-fidelity settings and more operationally relevant objectives. High-priority directions include (i) improved reward structures that emphasize terminal performance while retaining informative dense feedback, (ii) incorporation of time-varying or variable step sizes and time-of-flight optimization and (iii) the completion of a pretraining stack for rendezvous trajectories. Additional further objectives are expansion to three-dimensional state space, adding multi-body dynamics, including uncertainty and navigation error. These extensions would help determine the extent to which hybrid supervised–RL pipelines can scale beyond benchmark problems and serve as practical complements to classical optimal control in future low-thrust mission design and autonomous guidance applications.

Bibliography

[1] Mykel J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[2] OpenAI. Part 2: Kinds of RL algorithms, 2018. Spinning Up in Deep RL documentation. Accessed: 2026-01-23.

[3] Yanis Sidhoum and Kenshiro Oguri. On the performance of different smoothing methods for indirect low-thrust trajectory optimization. In *AAS/AIAA Astrodynamics Specialist Conference*, pages 207–227, Big Sky, MT, 2023. AIAA.

[4] ai-solutions, Inc. Targeting, available at: `https://ai-solutions.com/_help_Files/targeting.htm`. Accessed: 2025-05-08.

[5] Nathan L. Parrish and Daniel J. Scheeres. Optimal low-thrust trajectory correction with neural networks. In *AAS Paper 18-397*, 2018. Paper number: AAS 18-397.

[6] Frank Laipert Ari Rubinsztejn, Rohan Sood. Neural network optimal control in astrodynamics: Application to the missed thrust problem. In *Acta Astronautica*, pages 192–203. publisher=Elsevier, 2020.

[7] Howell LaFarge, Miller and Linares. Autonomous closed-loop guidance using reinforcement learning in a low-thrust, multi-body dynamical environment. *Acta Astronautica*, 186:1–23, 09 2021.

[8] Denis Auroux. Session 39: Statement of lagrange multipliers and example, 2010.

[9] L. Pontryagin. *Mathematical Theory of Optimal Processes*. Interscience, New York, 1962.

[10] Bruce A. Conway, editor. *Spacecraft Trajectory Optimization*. Cambridge Aerospace Series. Cambridge University Press, Cambridge, United Kingdom, 2010.

[11] Mischa Kim. *Continuous Low-Thrust Trajectory Optimization: Techniques and Applications*. Ph.d. dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA, April 2005.

[12] Jon A. Sims and Steve N. Flanagan. Preliminary design of low-thrust interplanetary missions. Technical report, Jet Propulsion Laboratory, California Institute of Technology, 1997. NASA NTRS accession 20000057422.

[13] Brittanny V. Holden, Shan He, and Anil V. Rao. Minimum-time earth-to-mars interplanetary orbit transfer using adaptive gaussian quadrature collocation. arXiv preprint, 2021. arXiv:2012.04116.

[14] Nathan Luis Olin Parrish. *Low Thrust Trajectory Optimization in Cislunar and Translunar Space*. PhD thesis, University of Colorado at Boulder, 2018.

[15] Alex Pascarella, Robyn Woollands, Etienne Pellegrini, Marc Sanchez-Net, and Joshua Vander Hook. Low-thrust trajectory optimization for the solar system pony express. Technical Report AAS 22-015, Jet Propulsion Laboratory, California Institute of Technology, 2022. AAS Paper 22-015.

[16] Yazhe Meng, Hao Zhang, and Yang Gao. Low-thrust minimum-fuel trajectory optimization using multiple shooting augmented by analytical derivatives. *Journal of Guidance, Control, and Dynamics*, 42(3):662–677, 2019.

[17] Arthur E. Bryson and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Taylor & Francis, New York, 1975.

[18] R. Bertrand and R. Epenoy. New smoothing techniques for solving bang-bang optimal control problems—numerical results and statistical interpretation. *Optimal Control Applications and Methods*, 23:171–197, 2002.

[19] Ehsan Taheri, Ilya Kolmanovsky, and Ella Atkins. Enhanced smoothing technique for indirect optimization of minimum-fuel low-thrust trajectories. *Journal of Guidance, Control, and Dynamics*, 39:2500–2511, 2016.

[20] John C. Parrish, Jeffrey S. Parker, Steven P. Hughes, and Jeannette Heiligers. Low-thrust transfers from distant retrograde orbits to l2 halo orbits in the earth-moon system, March 2016. Manuscript/PDF (publication venue not specified in the uploaded copy).

[21] J. E. Hecht and N. Botta. Particle swarm optimization-based co-state initialization for low-thrust minimum-fuel trajectory optimization. *Acta Astronautica*, 211:416–430, 2023.

[22] Bilal Sidhoum and Makoto Oguri. Pontryagin-bellman differential dynamic programming for low-thrust trajectory optimization with path constraints, 2025.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. Accessed 2026-01-01.

[24] Martin T. Hagan, Howard B. Demuth, and Orlando De Jesús. An introduction to the use of neural networks in control systems. *International Journal of Robust and Nonlinear Control*, 12(11):959–985, 2002.

[25] J. Li, Y. Jia, and J. Zhang. Deep-learning-based trajectory optimization: A survey and a beyond perspective. *Journal of Systems Architecture*, 126:102505, 2022.

[26] W. Li, H. Huang, and F. Peng. Deep networks approximators of optimal low-thrust multi-impulse costs in multi-target missions, 2019.

[27] G. Viavattene and M. Ceriotti. Artificial neural networks for multiple nea rendezvous missions with continuous thrust. *Journal of Guidance, Control, and Dynamics*, 2021.

[28] Dario Izzo and Ekin Ozturk. Real-time guidance for low-thrust transfers using deep neural networks. *Journal of Guidance, Control, and Dynamics*, 44:1–13, 01 2021.

[29] Nathan L. Parrish and Daniel J. Scheeres. Low-thrust trajectory optimization with no initial guess. In *Proceedings of the 26th International Symposium on Space Flight Dynamics (ISSFD)*, Matsuyama, Japan, 2017. Conference paper.

[30] Massimo Tipaldi, Raffaele Iervolino, and Paolo Roberto Massenio. Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges. *Annual Reviews in Control*, 54:1–23, 2022.

[31] Harry Holt, Roberto Armellin, Andrea Scorsoglio, and Roberto Furfaro. Low-thrust trajectory design using closed-loop feedback-driven control laws and state-dependent parameters. In *AIAA Scitech 2020 Forum*, Orlando, FL, January 2020.

[32] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[33] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. 2018.

[34] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. 2020.

[35] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. 2021. NeurIPS 2021 Spotlight.

[36] Roger R. Bate, Donald D. Mueller, and Jerry E. White. *Fundamentals of Astrodynamics*. Dover Publications, 1971.

[37] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Dover Publications, Mineola, NY, 2004. Reprint of the 1970 Prentice-Hall edition.

[38] Derek F. Lawden. *Optimal Trajectories for Space Navigation*. Butterworths, London, 1963.

[39] John E. Prussing. Primer vector theory and applications. In Bruce A. Conway, editor, *Spacecraft Trajectory Optimization*, pages 16–36. Cambridge University Press, Cambridge, 2010.

[40] Régis Bertrand and Richard Epenoy. New smoothing techniques for solving bang-bang optimal control problems - numerical results and statistical interpretation. *Optimal Control Applications and Methods*, 23:171 – 197, 07 2002.

[41] Ryuhei Okumura, Dapeng Cai, and Takashi Gyoshin Nitta. Transversality conditions for infinite horizon optimization problems: Three additional assumptions. *ROMAI Journal*, 5(1):105–112, 2009. See p. 107, Eq. (5) for the finite-horizon case: free terminal state $\Rightarrow \lambda(T) = 0$.

[42] F. Jiang, H. Baoyin, and J. Li. Practical techniques for low-thrust trajectory optimization with homotopic approach. *Journal of Guidance, Control, and Dynamics*, 35(1):245–258, 2012.

[43] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 2 edition, 2018.

[44] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[45] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.

[46] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12:1008–1014, 2000.

[47] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 2018.

[48] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

[49] F. Laipert, A. Nicholas, R. Woolley, and R. Lock. Hybrid chemical–electric trajectories for a Mars sample return orbiter. In *AAS/AIAA Space Flight Mechanics Meeting*, number AAS 19-345, pages 1–8, 2019.

[50] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[51] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[52] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[53] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. Submitted 12/20; Revised 10/21; Published 11/21.