(Preprint) AAS 25-831

RISK-SENSITIVE REINFORCEMENT LEARNING FOR DESIGNING ROBUST LOW-THRUST INTERPLANETARY TRAJECTORIES

Aarun Srinivas* **and John R. Martin†**

In recent years, small spacecraft have been increasingly proposed for interplanetary missions due to their cost-effectiveness, rapid development cycles, and ability to perform complex tasks comparable to larger spacecraft. However, these benefits come with trade-offs, as limited budgets often necessitate using components with low technological readiness, increasing the risk of control execution errors occurring from misaligned thrusters, actuator noise, and missed-thrust events. Recently, Reinforcement learning (RL) has emerged as a promising approach for designing robust trajectories that account for these errors. However, existing methods depend on prior knowledge of these errors to construct training simulations, restricting their ability to generalize to unforeseen anomalies. To overcome this limitation, we propose using Risk-Sensitive Reinforcement Learning (RSRL) to train policies that remain robust to control execution errors without requiring prior knowledge of their exact nature, enhancing practicality for real-world missions. In particular, we build our RSRL algorithm on top of the Proximal Policy Optimization (PPO) RL algorithm by replacing its risk-neutral objective with the risk-sensitive exponential criterion. We evaluate our RSRL algorithm, RS-PPO, by comparing its performance against PPO in an interplanetary transfer from Earth to Mars, where both are trained in an error-free environment but tested under various control execution errors.

INTRODUCTION

The use of small spacecraft (SmallSats) in interplanetary missions has gained significant traction in recent years, as evidenced by upcoming missions such as ESA's M-ARGO[1] and NASA's INSPIRE.[2] This surge in adoption is largely driven by the success of previous missions like PROCYON,[3] developed by the University of Tokyo and JAXA, and NASA's Mars Cube One,[4] both of which delivered valuable scientific results while being built at low cost and within short timeframes. However, despite their successes, SmallSats frequently encounter unexpected control execution errors, as their limited budgets often necessitate the use of components with low technological readiness levels (TRL). For example, PROCYON experienced a malfunction in its main thruster, preventing it from executing its planned flyby of asteroid 2000 DP107. Similarly, NASA's Lunar Flashlight CubeSat[5] suffered underperformance in three of its four thrusters, while LunaH-Map[6] was hindered by a stuck valve in its electric thruster. These issues significantly compromised both missions' ability to reach lunar orbit.

In this paper, we aim to address the shortfalls of SmallSats by employing Risk-Sensitive Reinforcement Learning (RSRL) algorithms to design robust low-thrust interplanetary trajectories. We focus on the low-thrust setting, as low-thrust electric propulsion has been a key technology

*Graduate Research Assistant, Aerospace Engineering, University of Maryland - College Park.
†Assistant Professor, Aerospace Engineering, University of Maryland - College Park.

in enabling SmallSats to perform interplanetary missions with significantly lower specific propellant consumption. As a result, potential control execution errors that could occur during low-thrust interplanetary transfers with low-TRL components may arise from actuator noise, thruster misalignments, and missed-thrust events (MTEs). Figure 1 illustrates the expected performance of our RSRL approach compared to traditional RL methods in the presence of such control execution errors.

To explain our approach and findings, we have structured the paper as follows. We begin by providing an overview of state-of-the-art approaches in optimal control and RL that have been used to design low-thrust interplanetary trajectories. We then describe how the interplanetary transfer problem is modeled as a Markov Decision Process (MDP), allowing us to solve it using RL algorithms like Proximal Policy Optimization (PPO), which is introduced in the following section. Next, we explore Risk-Sensitive RL at a high level, before focusing specifically on the exponential criterion, which we aim to optimize for designing robust low-thrust interplanetary trajectories. We then outline the implementation details, including the construction of the training and testing environments, the development of our RSRL algorithm, and the identification of the hyperparameters used to optimize both the PPO and RSRL algorithms. The effectiveness of our approach is then demonstrated in our results section, which highlights the superior robustness of the RSRL method in handling control execution errors compared to PPO. Finally, we conclude with a summary of key findings, their broader implications, and suggestions for future work.

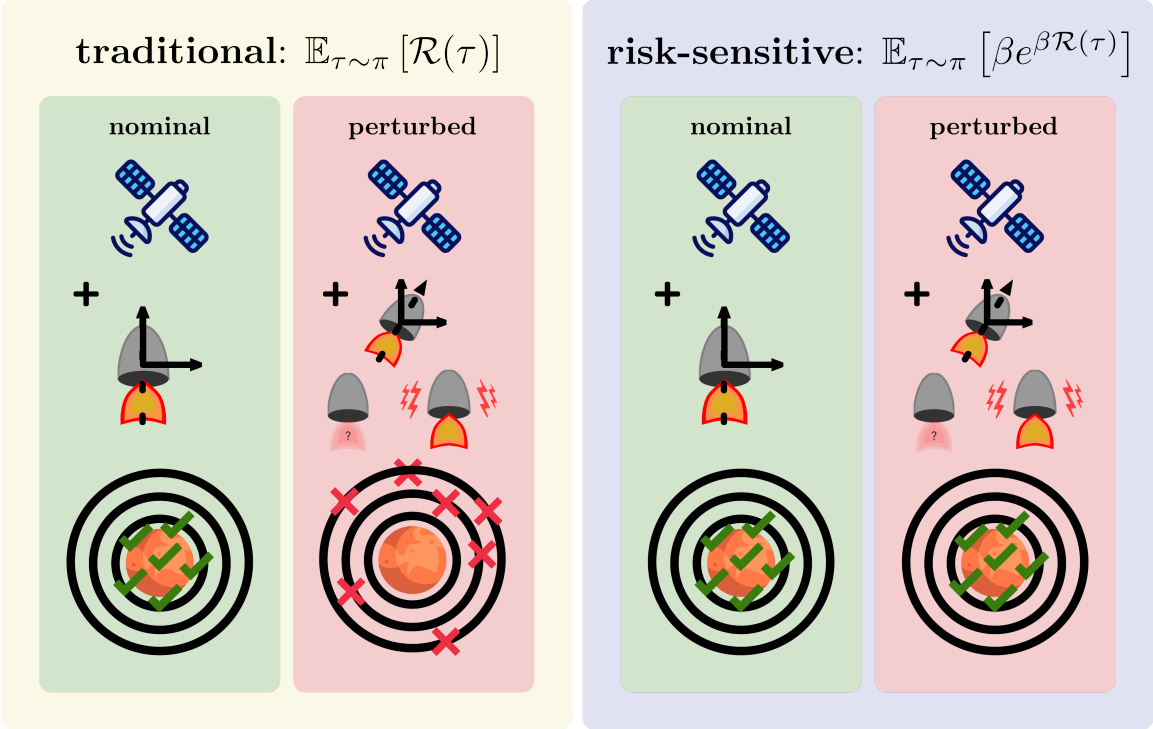


Figure 1: Comparison of traditional and risk-sensitive reinforcement learning for performing an interplanetary transfer to Mars, under both nominal conditions and in the presence of control execution errors caused by actuator noise, misaligned thrusters, and missed-thrust events.

RELATED WORK

Optimal Control

Historically, optimal control methods—both indirect approaches based on Pontryagin's principle[7] and direct methods utilizing collocation techniques[8,9]—have excelled at designing time-optimal or minimum-propellant low-thrust trajectories. However, these methods rely on the assumption that the spacecraft's operational environment is completely known and static, a condition that is not always met. To address this limitation, convex-optimization techniques have been introduced for low-thrust trajectory design.[10,11] These techniques, which follow the Model Predictive Control (MPC) framework, are computationally efficient and therefore suitable for onboard spacecraft guidance. MPC operates by iteratively solving an optimal control problem (OCP) over a finite horizon, where at each time step the controller computes an optimal action sequence, applies the first action, and updates the OCP's initial conditions using the latest system measurements. This process repeats in a receding-horizon fashion until the control objective is achieved. This iterative approach enables the spacecraft to continuously adapt its trajectory at every step, making it an appealing method for guidance.[12] However, MPC requires that both the objective and the dynamics of the OCP be convex, which is often not the case in low-thrust trajectory design problems. As a result, sequential convex programming methods must be employed to render the OCP solvable, resulting in a slower optimization process.[13,14] This slow adaptation can hinder the algorithm's ability to respond to control execution errors in real time, making it less effective for low-thrust trajectory design in SmallSat interplanetary missions.

Deep Learning

Over the past few years, deep learning techniques have gained popularity for trajectory design due to their ability to overcome the limitations faced by traditional optimal control methods. One such method is behavior cloning (BC), which involves utilizing a dataset of expert-generated trajectories to train neural networks for optimal trajectory design, has demonstrated utility in designing interplanetary transfers[15] as well as optimal asteroid landing trajectories.[16] Unfortunately, this method's performance decreases rapidly when it is tasked with designing trajectories that lie outside the expert-generated dataset it was trained on. In other words, BC is unable to generalize. This issue becomes particularly problematic when BC, often trained on trajectories derived in deterministic settings, is deployed in stochastic environments, where randomness can quickly cause deviations from the learned trajectories. Therefore, it is an inadequate approach for designing low-thrust trajectories that can deal with unanticipated control execution errors.

Reinforcement Learning (RL), on the other hand, does not require expert-generated demonstrations and can instead learn optimal policies through direct interactions with its environment, which is modeled as a MDP. In particular, RL agents achieve this by balancing exploration, which allows them to uncover new behaviors, with exploitation, where they capitalize on proven strategies to enhance performance. To distinguish beneficial actions that drive progress from detrimental ones that violate constraints, the agent receives scalar rewards from the MDP as feedback, which guides its learning process. This approach of learning from experience has allowed RL to achieve success in various spacecraft applications, including low-thrust trajectory design,[17] landing guidance,[18] cis-lunar trajectory optimization,[19–21] and rendezvous and docking maneuvers.[22] Furthermore, RL-based methods for robust trajectory design have demonstrated the ability to develop policies that can accommodate control execution errors.[23] However, existing work assumes that both the type

and magnitude of control execution errors are known beforehand and can therefore be simulated. This is often not the case, as evidenced by various SmallSats like PROCYON, Lunar Flashlight, and LunaH-Map experiencing unforeseen errors, highlighting the need for RL agents to generalize to novel and unexpected scenarios. Unfortunately, the RL algorithms that have been successfully applied in aerospace are often limited in such situations, as they can be sensitive to initial conditions, prone to instability, and lack the robustness needed to effectively handle unanticipated errors.

PROBLEM STATEMENT

The primary objective of this paper is to investigate how RSRL algorithms can be leveraged to design low-thrust interplanetary trajectories that are robust to control execution errors. To assess the effectiveness of our approach and facilitate comparisons with existing research, we focus on a three-dimensional, time-fixed, minimum-fuel Earth-Mars rendezvous mission. In this scenario, the spacecraft departs from Earth and executes a series of maneuvers with the goal of matching Mars' position and velocity while minimizing fuel consumption. Initially, we consider a simplified model where the spacecraft is influenced solely by the gravitational forces of the Earth, Mars, and the Sun. To assess the robustness of the trained RSRL policies compared to RL baselines, however, we simulate control execution errors caused by factors such as actuator noise, thruster misalignments, and MTEs. Table 1 summarizes key experimental parameters, including the total number of simulation steps N, transfer time t_f, initial spacecraft mass m_0, and engine characteristics (maximum thrust T_{max} and exhaust velocity u_{eq}). It also provides the initial position $\mathbf{r}_\oplus$ and velocity $\mathbf{v}_\oplus$ of Earth, the final position $\mathbf{r}_\sigma$ and velocity $\mathbf{v}_\sigma$ of Mars, and the gravitational parameters $\mu_\oplus$, μ_σ, and $\mu_\odot$ for Earth, Mars, and the Sun, respectively. In all of our simulations, the spacecraft's position, velocity, and mass are non-dimensionalized by normalizing with respect to the Earth–Sun mean distance $\bar{r} = 1.496 \times 10^{11}$ m, the corresponding orbital velocity $\bar{v} = \sqrt{\mu_\odot / \bar{r}}$, and the initial spacecraft mass $\bar{m} = m_0$.

Table 1: Problem Data

Variable	Value
N	40
t_f, days	358.79
T_{max}, N	0.5
u_{eq}, m/s	19,613.3
m_0, kg	1000
$\mu_\odot$, m^3/s^2	1.327×10^{20}
$\mu_\oplus$, m^3/s^2	3.986×10^{14}
μ_σ, m^3/s^2	4.282×10^{13}
$\mathbf{r}_\oplus$, m	$[-1.407 \times 10^{11}, -5.161 \times 10^{10}, 9.8 \times 10^{5}]^T$
$\mathbf{v}_\oplus$, m/s	$[9.775 \times 10^{3}, -2.808 \times 10^{4}, 0.434]^T$
$\mathbf{r}_\sigma$, m	$[-1.727 \times 10^{11}, 1.770 \times 10^{11}, 7.949 \times 10^{9}]^T$
$\mathbf{v}_\sigma$, m/s	$[-1.643 \times 10^{4}, -1.486 \times 10^{4}, 0.921]^T$

Markov Decision Process

In this section, we will briefly introduce the concept of a MDP, which is a mathematical framework used to model decision-making problems where an agent learns to accomplish a specific goal

by interacting with its environment.[24] It is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, p_0)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition function, $r : S \to \mathbb{R}$ is the reward function, $p_0 : \mathcal{S} \to \mathbb{R}$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor. The agent's objective, when solving an MDP, is to find the optimal policy π^* that maximizes the expected return $\mathcal{R}$ or discounted sum of rewards:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\mathcal{R}(\tau)\right] = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=0}^{N-1} \gamma^k r(\mathbf{s}_k)\right], \quad \text{where} \quad \tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots) \tag{1}$$

$$\mathbf{s}_0 \sim \rho_0(\mathbf{s}_0), \quad \mathbf{a}_k \sim \pi(\mathbf{a}_k|\mathbf{s}_k), \quad \mathbf{s}_{k+1} \sim P(\mathbf{s}_{k+1}|\mathbf{s}_k, \mathbf{a}_k)$$

Formulating the Earth-to-Mars Transfer Problem as a Markov Decision Process

We will now outline how the problem of finding an optimal Earth-to-Mars transfer is formulated as an MDP. The first step of this formulation is defining the MDP's state space, which encompasses all possible positions and velocities in Cartesian coordinates as well as masses that the spacecraft can assume at time $t_k = k\frac{t_f}{N}, k \in [0, N]$.

$$\mathbf{s}_k = \begin{bmatrix}\mathbf{r}_k^T & \mathbf{v}_k^T & m_k\end{bmatrix}^T \in \mathbb{R}^7 \tag{2}$$

We then adopt the Sims-Flanagan model to approximate a low-thrust trajectory as a sequence of ballistic arcs connected by impulsive ΔVs. With this model, we can determine the maximum ΔV the agent can take at time t_k using the spacecraft engine's maximum thrust T_{max}:

$$\Delta V_{max,k} = \frac{T_{max}}{m_k}\frac{t_f}{N} \tag{3}$$

This allows us to bound the MDP's action space within a ball of radius equal to $\Delta V_{max,k}$:

$$\mathbf{a}_k \in \{\Delta V_{\text{max},k} \cdot \mathbf{v} \mid \mathbf{v} \in \mathbb{B}^3\} \tag{4}$$

Since we operate in a three-body environment, our MDP's transition function is deterministic but lacks an analytic form. Therefore, we define our transition function f using the n-body equations of motion, where the velocity $\mathbf{v}_k$ is first updated instantaneously by $\Delta\mathbf{V}_k$ before numerical integration. Additionally, we apply the Tsiolkovsky equation to update the spacecraft's mass, ensuring an accurate transition from t_k to t_{k+1}:

$$\begin{bmatrix}\mathbf{r}_{k+1}\\ \mathbf{v}_{k+1}\\ m_{k+1}\end{bmatrix} = f(\mathbf{r}_k, \hat{\mathbf{v}}_k, m_k, \Delta\mathbf{V}_k) = \begin{bmatrix}\mathbf{r}_k + \int_{t_k}^{t_{k+1}} \hat{\mathbf{v}}_k\, dt\\ \hat{\mathbf{v}}_k + \int_{t_k}^{t_{k+1}} \left(-\mu_\odot \frac{\mathbf{r}-\mathbf{r}_\odot}{|\mathbf{r}-\mathbf{r}_\odot|^3} - \mu_\oplus \frac{\mathbf{r}-\mathbf{r}_\oplus}{|\mathbf{r}-\mathbf{r}_\oplus|^3} - \mu_\male \frac{\mathbf{r}-\mathbf{r}_\male}{|\mathbf{r}-\mathbf{r}_\male|^3}\right) dt\\ m_k \exp\left(\frac{-|\Delta\mathbf{V}_k|}{u_{eq}}\right)\end{bmatrix} \tag{5}$$

$$\text{where} \quad \hat{\mathbf{v}}_k = \mathbf{v}_k + \Delta\mathbf{V}_k$$

When computing the $\Delta\mathbf{V}$ at time t_f, we force the spacecraft to execute a maneuver that matches its velocity to that of Mars:

$$\Delta \mathbf{V}_N = \min\left(|\mathbf{v}_{\mathcal{O}^{\prime}} - \mathbf{v}_N|, \Delta V_{\max,N}\right) \frac{\mathbf{v}_{\mathcal{O}^{\prime}} - \mathbf{v}_N}{|\mathbf{v}_{\mathcal{O}^{\prime}} - \mathbf{v}_N|} \tag{6}$$

This allows us to express the final state of the spacecraft as:

$$\mathbf{r}_f = \mathbf{r}_N \tag{7}$$

$$\mathbf{v}_f = \mathbf{v}_N + \Delta \mathbf{V}_N \tag{8}$$

$$m_f = m_N \exp\left(\frac{-|\Delta \mathbf{V}_N|}{u_{eq}}\right) \tag{9}$$

To evaluate the robustness of the designed low-thrust trajectories, we introduce control execution errors into the Earth-to-Mars MDP by transforming the agent's selection action $\mathbf{a}_k$ into $\hat{\mathbf{a}}_k$ prior to feeding it into our transition function, resulting in a new transition function $\hat{f}$:

$$\hat{f}(\mathbf{r}_k, \hat{\mathbf{v}}_k, m_k, \mathbf{a}_k) = f(\mathbf{r}_k, \hat{\mathbf{v}}_k, m_k, \hat{\mathbf{a}}_k) \tag{10}$$

In our experiment, we consider control execution errors arising from actuator noise, thruster misalignments, or MTEs. For modeling actuator noise, we add Gaussian noise to the agent's selected action:

$$\hat{\mathbf{a}}_k = \mathbf{a}_k + \delta \mathbf{a}_k \tag{11}$$

$$\text{where} \quad \delta \mathbf{a}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{U}_{a,k}) \in \mathbb{R}^3, \quad \boldsymbol{U}_{a,k} = \sigma_a I_3$$

To model thruster misalignment, we sample a unit vector $\mathbf{e}$ uniformly from the unit sphere at the start of each episode. This vector, combined with a predefined rotation angle ϕ, defines the Euler parameter vector $\boldsymbol{\beta}(\phi, \mathbf{e})$. The corresponding direction cosine matrix $\boldsymbol{R}(\boldsymbol{\beta})$ is then used to rotate the agent's action at each timestep.

$$\hat{\mathbf{a}}_k = [\boldsymbol{R}(\boldsymbol{\beta}(\phi, \mathbf{e})]\mathbf{a}_k \tag{12}$$

$$\text{where} \quad \mathbf{e} \sim \mathcal{U}(\mathbb{S}^3)$$

Finally, MTEs are modeled as a complete loss of thrust, nullifying any action selected by the agent when they occur. To simulate MTEs, we begin each episode by sampling a trigger time $t_{\hat{k}}$ uniformly from the interval $[0, N)$, at which point the MTE is initiated, setting $\mathbf{a}_k = 0$. After activation, the MTE persists at each subsequent timestep with probability p_{mte}, and can last for at most n_{mte} consecutive steps. Once the system recovers, however, the MTE does not occur for the remainder of the episode. The specific parameters used to model all control execution errors are summarized in Table 2.

Table 2: Uncertainty Model Parameters

σ_a, m/s	ϕ, deg	p_{mte}	n_{mte}
25	5	0.1	3

To promote fuel efficiency and ensure satisfaction of terminal constraints during the Earth-to-Mars transfer, the reward function penalizes the agent at each timestep for propellant usage and applies a large terminal penalty based on position and velocity constraint violations, weighted by penalty factors $\lambda_1 = 10$, $\lambda_2 = 100$, and $\lambda_3 = 100$.

$$r_k = \begin{cases} -\lambda_1 \cdot (m_{k-1} - m_k) & \text{if } k < N, \\ -\lambda_1 \cdot (m_f - m_k) - \lambda_2 \cdot \ln\left(\lambda_3 \cdot \left(\frac{|r_f - r_{\sigma}|}{|r_{\sigma}|} + \frac{|v_f - v_{\sigma}|}{|v_{\sigma}|}\right)\right) & \text{if } k = N \end{cases} \tag{13}$$

We use the natural logarithm in the terminal penalty to structure the reward such that large violations lead to significant negative rewards, while smaller violations yield progressively larger positive rewards. This transition from negative to positive rewards near the target incentivizes the agent to quickly match Mars's position and velocity, accelerating convergence during training.

REINFORCEMENT LEARNING

To establish a RL benchmark for solving the Earth-to-Mars transfer problem, we utilize the Proximal Policy Optimization (PPO) RL algorithm. PPO is a model-free, policy-gradient actor-critic algorithm known for its outstanding performance in continuous and high-dimensional control tasks, making it a preferred choice for RL applications in the astrodynamics community.[25] As a model-free algorithm, PPO does not rely on a predefined model of the environment's dynamics, instead learning a policy π directly from real-world samples. This policy is parameterized by a deep neural network (DNN) with parameters θ, denoted as π_θ to highlight its dependence on the network's parameters. To manage complex environments with large and continuous state and action spaces, the DNN is often composed of multiple layers, with each layer consisting of neurons that compute a weighted sum of inputs from the previous layer and apply a nonlinear activation function. In the case of a deterministic policy, the final layer of the DNN directly outputs the action. For stochastic policies, such as those learned by PPO, the final layer produces either a categorical distribution for discrete action spaces or the mean and variance of a Gaussian distribution for continuous action spaces, from which actions are sampled accordingly.

As a policy-gradient method, PPO aims to learn a stochastic policy by performing the following search for θ^*:

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{k=0}^{N-1} \gamma^k r(\mathbf{s}_k)\right] \tag{14}$$

This search is conducted using stochastic gradient ascent of the form $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$ to update the policy in the direction that maximizes the objective. Here, α is the learning rate and the policy gradient, $\nabla_\theta J(\theta)$, is computed using the policy gradient theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{k=0}^{N-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_k \mid \mathbf{s}_k) Q^{\pi_\theta}(\mathbf{s}_k, \mathbf{a}_k)\right] \tag{15}$$

where $Q^{\pi_\theta}(\mathbf{s}, \mathbf{a})$ represents the expected return from taking action a in state s and following policy π_θ:

$$Q^{\pi_\theta}(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{k'=k}^{N-1} \gamma^{k'-k} r_{k'} \mid \mathbf{s}_k = \mathbf{s}, \mathbf{a}_k = \mathbf{a} \right] \tag{16}$$

This expectation is typically estimated using one of two approaches. The first, the Monte Carlo method, computes the expectation based on rewards collected along sampled trajectories, yielding an unbiased but high-variance estimate. A more effective approach, however, is the Actor-Critic method, which utilizes a separate neural network, known as the critic, to learn an approximation of $Q^{\pi_\theta}(\mathbf{s}, \mathbf{a})$. This approach can be further improved by subtracting a baseline function from $Q^{\pi_\theta}(\mathbf{s}, \mathbf{a})$ to help increase training stability. A common choice for this baseline is the state value function $V^{\pi_\theta}(\mathbf{s})$, leading to the definition of the advantage function $A^{\pi_\theta}(\mathbf{s})$, which quantifies the relative benefit of taking action a in state s compared to following the current policy.

$$V^{\pi_\theta}(\mathbf{s}) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{k'=k}^{N-1} \gamma^{k'-k} r_{k'} \mid \mathbf{s}_k = \mathbf{s} \right] \tag{17}$$

$$A^{\pi_\theta}(\mathbf{s}, \mathbf{a}) = Q^{\pi_\theta}(\mathbf{s}, \mathbf{a}) - V^{\pi_\theta}(\mathbf{s}) \tag{18}$$

This approach forms the basis of the Advantage Actor-Critic (A2C) framework. In practice, the advantage values are computed using Generalized Advantage Estimation (GAE), where the parameter λ governs the bias-variance trade-off:[26]

$$\hat{A}_k = \sum_{k'=k}^{N-1} (\gamma\lambda)^{k'-k} \delta_{k'} \tag{19}$$

$$\text{where} \quad \delta_k = r_k + \gamma V^{\pi_\theta}(\mathbf{s}_{k+1}) - V^{\pi_\theta}(\mathbf{s}_k)$$

are unbiased estimates, as $\mathbb{E}_{\tau \sim \pi_\theta}[\delta_k] = A^{\pi_\theta}(\mathbf{s}_k, \mathbf{a}_k)$. This reformulation allows for the policy gradient to be evaluated more efficiently by reducing variance in gradient estimates, thereby enhancing training efficiency and stability.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{k=0}^{N-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_k | \mathbf{s}_k) \hat{A}_k \right] \tag{20}$$

PPO further improves upon the A2C framework by introducing a clipped surrogate objective function, which restricts the updated policy to remain within a small range ϵ. This prevents excessively large updates, which can often lead to the problem of catastrophic forgetting in RL. The clipped surrogate objective function is defined as:

$$J^{clip}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{k=0}^{N-1} \min \left(\tilde{r}_k \hat{A}_k,\ \text{clip}(\tilde{r}_k, 1-\epsilon, 1+\epsilon) \hat{A}_k \right) \right] \tag{21}$$

$$\text{where} \quad \tilde{r}_k(\pi_\theta) = \frac{\pi_\theta(\mathbf{a}_k | \mathbf{s}_k)}{\pi_\theta^{\text{old}}(\mathbf{a}_k | \mathbf{s}_k)}$$

is a ratio of the new and old policies.

As of now, the objective functions and corresponding policy gradients discussed have focused solely on updating the actor. For updating the critic, stochastic gradient descent is utilized to minimize the mean squared error objective H and thereby learn the value function for the current policy:

$$H(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\frac{1}{N}\sum_{k=0}^{N-1}\frac{1}{2}\left(V^{\pi_\theta}(\mathbf{s}_k) - \sum_{k'=k}^{N-1}\gamma^{k'-k}r_{k'}\right)^2\right] \tag{22}$$

In addition to the actor and critic objectives, PPO also utilizes an entropy regularization term S to encourage exploration and prevent premature convergence:

$$S(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\frac{1}{N}\sum_{k=0}^{N-1}\mathbb{E}_{\mathbf{a}_k\sim\pi_\theta(\cdot|\mathbf{s}_k)}\left[-\log\pi_\theta(\mathbf{a}_k \mid \mathbf{s}_k)\right]\right] \tag{23}$$

These objectives are then combined into a single loss function by weighting them with hyperparameters c_1 and c_2, which control the contributions of the value function objective and entropy terms, respectively:

$$J^{ppo}(\theta) = J^{clip}(\theta) - c_1 H(\theta) + c_2 S(\theta) \tag{24}$$

This objective is optimized through an iterative learning process consisting of two phases. The first phase is the policy rollout, which involves using the current policy to gather data to fill a rollout buffer. This buffer temporarily stores the collected data for the second phase, known as the policy update, where the data is sampled with batch size n_b, used to evaluate the objective, and perform n_{opt} epochs of stochastic gradient ascent. PPO alternates between performing policy rollouts and policy updates until the total number of training steps T is reached.

RISK-SENSITIVE REINFORCEMENT LEARNING

Although optimizing the expected return has yielded promising results, it often leads to policies that are sensitive to initial conditions, prone to instability, and lack robustness. This issue stems from the expectation operator in the objective, which tends to overlook rare but significant trajectories. As a result, this objective does not account for risk and is considered risk-neutral. In contrast, RSRL approaches are less affected by these issues and therefore have gained increasing attention. There are two primary methods for introducing risk into the risk-neutral objective that reinforcement learning optimizes: incorporating risk as a constraint or embedding it explicitly into the objective function. In this paper, we focus on the latter approach and optimize the exponential criterion, where $\beta > 0$ corresponds to a risk-seeking objective, and $\beta < 0$ corresponds to a risk-averse objective:[27]

$$J_\beta(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\beta e^{\beta\mathcal{R}(\tau)}\right] \tag{25}$$

To understand why the exponential criterion incorporates risk into the objective function, we can take its Taylor expansion, which consists of an infinite sum of higher moments of the return with diminishing weights for small values of β:

$$\mathbb{E}_{\tau\sim\pi_\theta}[\beta e^{\beta R}] = \beta + \beta^2\mathbb{E}_{\tau\sim\pi_\theta}[\mathcal{R}(\tau)] + \frac{\beta^3}{2}\mathbb{E}_{\tau\sim\pi_\theta}[\mathcal{R}^2(\tau)] + \dots \tag{26}$$

This expansion allows us to see that optimizing the exponential criterion is analogous to optimizing the risk-neutral objective, with an additional bonus or penalty for high-variance returns depending on the sign of β. Additionally, as β approaches zero, the criterion converges to the risk-neutral objective as the higher-order terms drop out. The exponential criterion can also be interpreted as representing the worst-case return when $\beta < 0$ and the best-case return when $\beta > 0$.[28] This interpretation, in particular, is the reason we selected the exponential criterion over alternative risk measures. Since we aim to learn policies that perform well in worst-case scenarios, the risk-averse exponential criterion will be especially useful for our application.

To optimize this objective, we adopt a risk-sensitive actor-critic algorithm. In this framework, advantage values are computed as the difference between the exponential Bellman backup, $R_k^\beta(\mathbf{s}_k, \mathbf{a}_k)$ —analogous to $Q(\mathbf{s}_k, \mathbf{a}_k)$ in standard actor-critic methods—and a scaled version of the risk-sensitive value function $V_\beta^{\pi_\theta}$, denoted as $\bar{V}_\beta^{\pi_\theta}$, which is derived from the risk-sensitive objective $J_\beta(\theta)$:

$$V_\beta^{\pi_\theta}(\mathbf{s}_k) = \beta\mathbb{E}_{\tau\sim\pi_\theta}\left[\exp\left(\beta\sum_{k'=k}^{N-1}\gamma^{k'-k}r(\mathbf{s}_{k'}, \mathbf{a}_{k'})\right) \mid \mathbf{s}_k\right] \tag{27}$$

$$\bar{V}_\beta^{\pi_\theta}(\mathbf{s}_k) = \frac{1}{\beta}V_\beta^{\pi_\theta}(\mathbf{s}_k) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\exp\left(\beta\sum_{k'=k}^{N-1}\gamma^{k'-k}r(\mathbf{s}_{k'}, \mathbf{a}_{k'})\right) \mid \mathbf{s}_k\right] \tag{28}$$

$$R_k^\beta(\mathbf{s}_k, \mathbf{a}_k) = \exp\left(\beta r(\mathbf{s}_k, \mathbf{a}_k) + \gamma \ln \bar{V}_\beta^{\pi_\theta}(\mathbf{s}_{k+1})\right) \tag{29}$$

$$A_\beta^{\pi_\theta}(\mathbf{s}_k, \mathbf{a}_k) = R_k^\beta(\mathbf{s}_k, \mathbf{a}_k) - \bar{V}_\beta^{\pi_\theta}(\mathbf{s}_k) \tag{30}$$

Similar to A2C, these advantage values are then used to compute policy gradients with reduced variance, with the key exception that they are scaled by the risk parameter β, resulting in the policy gradient $\nabla J_\beta(\theta)$:

$$\nabla_\theta J_\beta(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\beta\sum_{k=0}^{N-1}\nabla_\theta \log \pi_\theta(\mathbf{a}_k|\mathbf{s}_k)A_\beta^{\pi_\theta}(\mathbf{s}_k, \mathbf{a}_k)\right] \tag{31}$$

To train the critic, we minimize the mean-squared error objective H_β:

$$H_\beta(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\frac{1}{N}\sum_{k=0}^{N-1}\frac{1}{2}\left(\bar{V}_\beta^{\pi_\theta}(\mathbf{s}_k) - R_k^\beta(\mathbf{s}_k, \mathbf{a}_k)\right)^2\right] \tag{32}$$

$$J^{risk}(\theta) = J_\beta(\theta) - c_1 H_\beta(\theta) \tag{33}$$

It is important to note that we do not include an entropy term S in our objective, as the risk-sensitive objective J^{risk} already serves as an upper bound on the standard RL objective J, which includes the entropy term.[29]

IMPLEMENTATION DETAILS

To develop our RSRL algorithm for designing robust low-thrust trajectories, we improve upon the risk-sensitive actor-critic method introduced in the earlier section. Specifically, we compute advantage values using generalized advantage estimation and integrate the clipping mechanism from PPO to mitigate the instability that can arise from using the exponential operator. This modified algorithm, which we call RS-PPO, is built upon the Proximal Policy Optimization (PPO) algorithm from Stable Baselines3—an open-source library providing high-quality implementations of RL algorithms in PyTorch—by replacing its standard actor-critic algorithm with the risk-sensitive variant proposed earlier. To evaluate RS-PPO, we compare its performance against the original PPO algorithm it was built upon, which serves as our baseline. Both algorithms are trained for a total of 1×10^6 timesteps to ensure a fair comparison. They also share the same underlying deep neural network architecture, consisting of separate actor and critic networks, each with two hidden layers. A summary of the network architecture, including the number of neurons per layer and activation functions, as well as hyperparameters is provided in Table 3 and Table 4.

Table 3: Network Architecture

	Policy Network	Value Network
Layer 1	64	64
Layer 2	64	64
Output	3	1
Activation	tanh	tanh

Table 4: Hyperparameters

Algorithm	Hyperparameters							
	γ	λ	α	ϵ	c_1	c_2	n_b	n_{opt}
PPO	0.9999	0.99	5×10^{-4}	0.3	0.5	4.75×10^{-8}	64	50
RS-PPO	0.9999	0.99	5×10^{-4}	0.3	0.5	-	64	50

For building our low-thrust interplanetary transfer environment, we utilize Basilisk, a high-performance spacecraft simulation framework, to construct our simulation engine. With Basilisk, we are able to model three-body dynamics as well as configure the spacecraft, Earth, and Mars such that they match the data in Table 1. This simulation engine serves as our transition function f, which is subsequently incorporated into a Gymanisum environment that defines the remaining components of the MDP. In addition to using Gymanisum to formalize the MDP, we emulate control execution errors arising from actuator noise, thruster misalignment, and MTEs within the Gymanisum environment prior to passing them into Basilisk, allowing us to utilize the same Basilisk environment throughout our entire experiment.

RESULTS

The results from training PPO and RS-PPO with varying risk parameters β on the nominal three-body environment are presented in Table 5, followed by an evaluation of the resulting policies under different control execution errors for comparison. To evaluate performance, we train each algorithm with five distinct random seeds. For every seed, we sample 1,000 trajectories from each environment to calculate average metrics that reflect the optimality of the resulting trajectories. These metrics

include the final spacecraft mass m_f, position constraint violation percentage $\Delta r_f/r_\sigma$, velocity constraint violation percentage $\Delta v_f/v_\sigma$, and the undiscounted return J.

Table 5: Robust Trajectory Overview

	Algorithm		Results							
		β	m_f, kg		$\Delta r_f/r_\sigma$, %		$\Delta v_f/v_\sigma$, %		J	
			mean	std	mean	std	mean	std	mean	std
Nominal	PPO	-	429.156	74.530	1.999	2.945	0.111	0.187	-10.018	118.015
	RS-PPO	-0.010	418.039	111.593	2.603	3.105	0.212	0.286	-23.338	154.937
		-0.005	**391.148**	**69.422**	**0.417**	**0.062**	**0.029**	**0.051**	**94.697**	**14.812**
		-0.001	386.117	53.468	0.426	0.101	0.018	0.017	94.072	22.076
		0.001	344.199	79.708	2.269	2.773	0.170	0.228	-20.336	141.755
		0.005	386.693	105.883	1.563	2.661	0.138	0.284	44.563	138.585
		0.010	340.687	86.029	2.670	3.261	0.336	0.440	-19.206	167.607
Actuator Noise	PPO	-	424.163	72.887	2.892	2.631	0.363	0.284	-81.283	84.382
	RS-PPO	-0.010	415.903	108.158	2.869	2.990	0.329	0.423	-59.099	127.717
		-0.005	**387.975**	**69.539**	**0.671**	**0.061**	**0.040**	**0.035**	**47.741**	**11.368**
		-0.001	382.925	52.333	0.805	0.235	0.028	0.021	36.386	24.322
		0.001	343.277	77.590	2.585	2.448	0.348	0.457	-63.479	110.947
		0.005	384.198	103.963	2.144	2.471	0.174	0.297	-34.216	95.986
		0.010	340.061	84.134	3.304	2.850	0.604	0.501	-97.083	104.315
Thruster Misalignment	PPO	-	425.882	72.674	4.048	2.468	1.110	0.981	-141.391	69.830
	RS-PPO	-0.010	420.068	104.077	3.399	3.132	0.882	0.675	-100.827	109.841
		-0.005	386.954	72.174	1.049	0.362	0.432	0.372	-10.393	39.502
		-0.001	**382.943**	**50.974**	**0.942**	**0.245**	**0.338**	**0.481**	**1.235**	**36.254**
		0.001	344.563	76.298	3.075	2.057	1.058	1.509	-109.766	91.129
		0.005	386.669	102.862	2.682	2.298	0.384	0.388	-83.330	80.842
		0.010	343.088	84.020	3.629	2.806	0.831	0.808	-126.748	83.638
Missed-Thrust Event	PPO	-	438.094	70.812	3.384	2.519	0.961	0.723	-103.049	76.089
	RS-PPO	-0.010	428.418	103.129	2.995	2.893	0.674	0.378	-78.010	115.923
		-0.005	**400.617**	**68.467**	**0.865**	**0.109**	**0.390**	**0.312**	**6.621**	**17.756**
		-0.001	400.884	49.287	1.318	0.561	0.461	0.422	-4.403	43.700
		0.001	356.692	70.079	3.234	2.368	0.792	0.504	-97.994	91.338
		0.005	400.220	98.697	2.633	2.176	0.826	0.285	-76.461	79.427
		0.010	358.416	83.304	4.058	2.644	0.983	0.292	-131.979	75.293

From Table 5, we can see that RS-PPO with a small negative risk parameter $\beta = -0.005$ or $\beta = -0.001$ consistently outperforms PPO across all settings. This is particularly evident in the perturbed environments, where optimizing the risk-averse objective allows RS-PPO to better prepare for worst-case scenarios caused by control execution errors. In addition to improved overall performance, RS-PPO demonstrates significantly smaller increases in position and velocity constraint violations compared to PPO. For example, in the actuator noise setting, PPO experiences a 1% increase in position violations and a 0.2% rise in velocity violations, while RS-PPO sees only a 0.2% increase in position violations and a 0.02% increase in velocity violations. A similar trend emerges under thruster misalignment, with PPO showing increases of 2% and 1% in position and velocity violations, respectively, while RS-PPO limits these to just 0.5%–0.6% for position and 0.3% for velocity. Even in the case of missed-thrust events, RS-PPO with mild risk aversion keeps position violations below 1%, further highlighting its robustness. In the nominal setting, RS-PPO also performs better, which may seem less obvious at first. We believe that by penalizing negative returns more strongly, the agent learns to avoid poor decisions early in training and is pushed to

explore more effectively. This leads to faster learning and better final performance even without any disturbances.

Interestingly, RS-PPO with a small positive risk parameter also outperforms PPO in all settings aside from the nominal case. This result is somewhat counterintuitive since risk-seeking policies are not typically associated with robustness to changes in dynamics. However, it is possible that the added risk encourages broader exploration in the nominal environment, allowing the agent to learn how to act in a wider range of states that PPO may not encounter until errors occur. These results suggest that incorporating risk, whether risk-averse or risk-seeking, can lead to more robust policies. At the same time, setting β too high or too low, such as $\beta = -0.01$ or $\beta = 0.01$, leads to performance across all metrics that is worse than or similar to that of PPO. This makes sense since excessive risk aversion can stifle exploration and prevent the discovery of optimal strategies, while overly risk-seeking behavior leads to reckless actions that accumulate large negative returns.

Although Table 5 clearly shows that RS-PPO with $\beta = -0.005$ or $\beta = -0.001$ outperforms standard PPO in terms of constraint violations, we note that these trajectories are not mass-optimal. In fact, PPO consistently conserves more mass across all scenarios. This trade-off arises from the structure of our reward function, which places greater emphasis on minimizing terminal position and velocity constraint violations than on fuel efficiency. As a result, RS-PPO with $\beta = -0.005$ is still able to achieve a higher return compared to the PPO baseline. We hypothesize that this issue can be addressed by either increasing the number of training timesteps, modifying the reward function to better balance fuel efficiency with constraint satisfaction, or a combination of both.

To further analyze the results in Table 5, we take the individual trajectories sampled from PPO, our best-performing risk-averse algorithm RS-PPO with $\beta = -0.005$, and our best-performing risk-seeking algorithm RS-PPO with $\beta = 0.005$ and aggregate their metrics across all scenarios and seeds into 100-bin histogram plots, as shown in Figure 2. These histogram plots provide a detailed view of the distribution of each metric across different risk parameter settings. In particular, these plots highlight the clear superiority of RS-PPO with $\beta = -0.005$ over its risk-seeking counterpart and PPO, as its return histogram follows a Gaussian-like distribution, while the other two exhibit greater overall spread and show an additional peak around large negative returns, likely caused by the thruster misalignment and missed-thrust event scenarios, which both methods struggled with. Furthermore, the multi-modal nature of the mass histograms across all algorithms suggests that the reward function permits trade-offs between final mass and constraint satisfaction—allowing similar returns to be achieved by sacrificing one for the other—especially given that the return histogram is largely Gaussian, aside from the distinct peak observed in the risk-seeking and risk-neutral cases.

To complement our numerical results, we also provide visualizations to compare the performance of the baseline PPO algorithm (Figure 3a) with our best-performing algorithm, RS-PPO with $\beta = -0.005$ (Figure 3b). For creating these visualizations, we sample 25 trajectories from each environment using policies trained on different seeds. These trajectories are then used to generate two plots, with the first displaying the entire transfer from Earth to Mars and the second offering a zoomed-in view near Mars. In the second plot, markers are placed at the endpoints of each trajectory to indicate velocity constraint violations. The size of each marker corresponds to the severity of the violation, with larger markers representing greater violations. These visualizations allow for an intuitive comparison of position and velocity constraint violations, offering additional insight into the performance differences between the algorithms.

Figure 3a demonstrates the limitations of PPO in generalizing to control execution errors. As

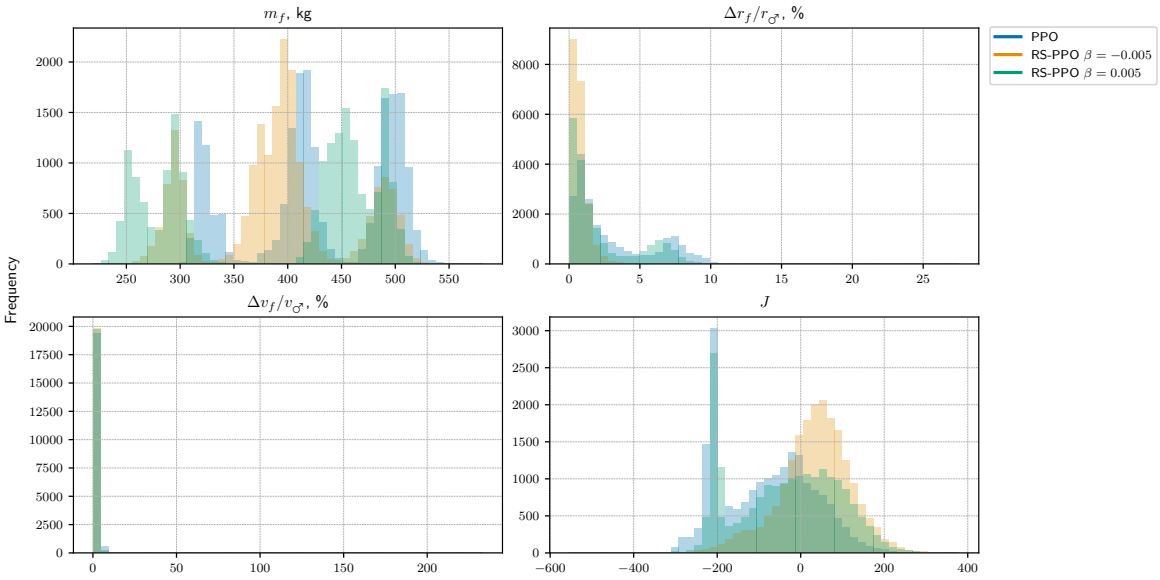


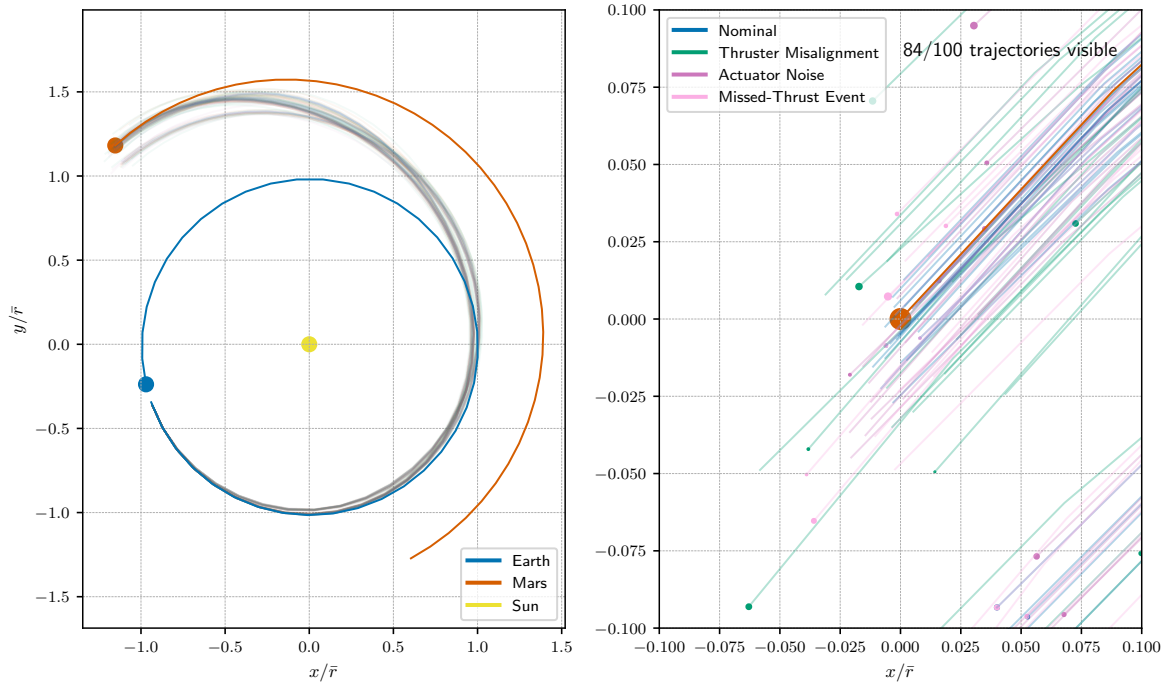
Figure 2: Histogram plots of performance metrics across 5,000 trajectories (1,000 per seed, five seeds) for PPO, RS-PPO with $\beta = -0.005$, and RS-PPO with $\beta = 0.005$.

shown in Table 5, the trajectories designed using PPO under nominal conditions remain noticeably closer to Mars, whereas those affected by execution errors fan out further from the target. We also observe that some of PPO's trajectories overshoot Mars, indicating that the agent does not account for control execution errors that could lead to overshooting. This behavior contrasts with a more cautious strategy that aims to undershoot, ensuring the spacecraft stays in a position where it can still reach Mars until the very end.

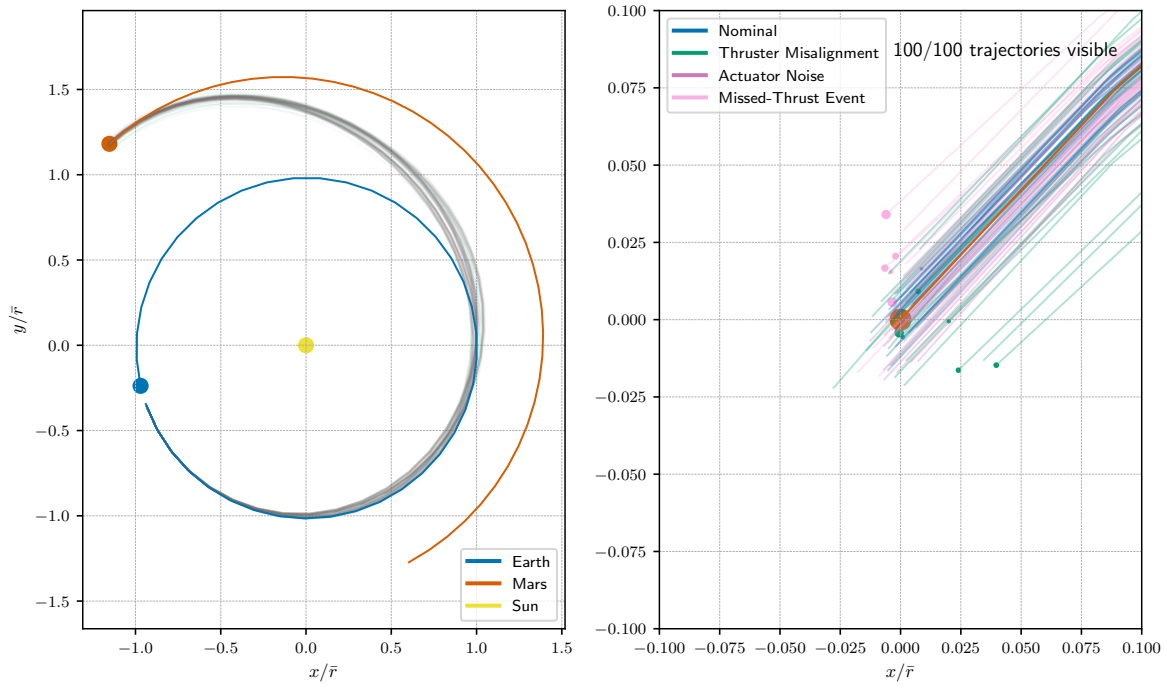
Figure 3b showcases the robustness of RS-PPO with $\beta = -0.005$ in the presence of control execution errors. As with PPO, we see that the trajectory behavior aligns closely with the results in Table 5, which indicate minimal position and velocity constraint violations. This is evident in both plots, where, unlike in Figure 3a, the trajectories do not fan out and instead remain tightly clustered around Mars, indicating consistently near-optimal performance across all settings. We also notice that the number of visible markers at the end of each trajectory is smaller compared to PPO, indicating smaller velocity constraint violations on average. Additionally, the overshooting observed is minimal and significantly less pronounced than in the PPO case. This suggests that RS-PPO with $\beta = -0.005$ learns a more cautious policy, likely favoring undershooting until the final moments in order to reliably reach Mars even in the presence of control execution errors.

CONCLUSION

In this paper, we propose the use of Risk-Sensitive Reinforcement Learning (RSRL) for designing low-thrust interplanetary trajectories that are robust to control execution errors. To assess the effectiveness of RSRL compared to standard RL algorithms in this setting, we focus on the problem of executing an interplanetary transfer from Earth to Mars, which we model as a Markov Decision Process (MDP). Then, we train both PPO and our risk-sensitive extension, RS-PPO, with varying risk parameters β, on the Earth-to-Mars MDP that we have constructed. Finally, we evaluate both our risk-neutral and risk-sensitive policies in the presence of control execution errors resulting from



(a) PPO



(b) RS-PPO with $\beta = -0.005$

Figure 3: Left: Earth-to-Mars trajectories generated using PPO and RS-PPO. Right: Zoomed-in view of the final approach, centered on Mars, which is located at the origin.

actuator noise, misaligned thrusters, and missed-thrust events to produce our results. Through our results, we observe that RS-PPO maintains performance significantly better than PPO when faced with previously unseen control execution errors. This demonstrates that the simple modification of replacing the standard RL objective with the exponential criterion in a state-of-the-art algorithm can result in more robust policies. While our approach is promising, several challenges must be overcome before it can be realistically used to design robust low-thrust trajectories onboard spacecraft. First, although our policies demonstrate some degree of robustness, no policy is universally robust, underscoring the need for retraining during deployment. Second, the optimal risk parameter β was selected through trial and error, without a principled method for tuning it, which is not feasible for real missions. To address both limitations, we plan to explore model-based RL methods, which are far more sample-efficient and therefore better suited for online adaptation, as well as approaches that allow β to be learned rather than treated as a fixed hyperparameter.

REFERENCES

[1] R. Walker, D. Koschny, C. Bramanti, I. Carnelli, E. Team, *et al.*, "Miniaturised Asteroid Remote Geophysical Observer (M-ARGO): a stand-alone deep space CubeSat system for low-cost science and exploration missions," *6th Interplanetary CubeSat Workshop, Cambridge, UK*, Vol. 30, 2017.

[2] A. T. Klesh, J. D. Baker, J. Bellardo, J. Castillo-Rogez, J. Cutler, L. Halatek, E. G. Lightsey, N. Murphy, and C. Raymond, "Inspire: Interplanetary nanospacecraft pathfinder in relevant environment," *AIAA SPACE 2013 Conference and Exposition*, 2013, p. 5323.

[3] S. Campagnola, N. Ozaki, Y. Sugimoto, C. H. Yam, H. Chen, Y. Kawabata, S. Ogura, B. Sarli, Y. Kawakatsu, R. Funase, *et al.*, "Low-thrust trajectory design and operations of PROCYON, the first deep-space micro-spacecraft," *25th International Symposium on Space Flight Dynamics*, Vol. 7, German Aerospace Center (DLR) Munich, Germany, 2015.

[4] S. Asmar and S. Matousek, "Mars Cube One (MarCO): the first planetary cubesat mission," *Proceedings of the Mars CubeSat/NanoSat Workshop, Pasadena, California, November*, Vol. 20, 2014, p. 21.

[5] M. Starr, M. Hauge, and E. G. Lightsey, "Shining a light on student-led mission operations: lessons learned from the Lunar Flashlight project," *AIAA SCITECH 2024 Forum*, 2024, p. 0822.

[6] R. Lane, C. Ryals, C. McLemore, and D. Hitt, "Nasa space launch system cubesats: first flight and future opportunities," 2023.

[7] L. Casalino and G. Colasurdo, "Optimization of Variable-Specific-Impulse Interplanetary Trajectories," *Journal of Guidance, Control, and Dynamics*, Vol. 27, 7 2004, pp. 678–684, 10.2514/1.11159.

[8] W. A. Scheel and B. A. Conway, "Optimization of very-low-thrust, many-revolution spacecraft trajectories," *Journal of Guidance, Control, and Dynamics*, Vol. 17, 11 1994, pp. 1185–1192, 10.2514/3.21331.

[9] K. F. Graham and A. V. Rao, "Minimum-Time Trajectory Optimization of Multiple Revolution Low-Thrust Earth-Orbit Transfers," *Journal of Spacecraft and Rockets*, Vol. 52, 5 2015, pp. 711–727, 10.2514/1.A33187.

[10] Z. Wang and M. J. Grant, "Optimization of Minimum-Time Low-Thrust Transfers Using Convex Programming," *Journal of Spacecraft and Rockets*, Vol. 55, 5 2018, pp. 586–598, 10.2514/1.A33995.

[11] Z. Wang and M. J. Grant, "Minimum-Fuel Low-Thrust Transfers for Spacecraft: A Convex Approach," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 54, 10 2018, pp. 2274–2290, 10.1109/TAES.2018.2812558.

[12] U. Eren, A. Prach, B. B. Koçer, S. V. Raković, E. Kayacan, and B. Açıkmeşe, "Model Predictive Control in Aerospace Systems: Current State and Opportunities," *Journal of Guidance, Control, and Dynamics*, Vol. 40, 7 2017, pp. 1541–1566, 10.2514/1.G002507.

[13] L. Federici, B. Benedikter, and A. Zavoli, "Machine Learning Techniques for Autonomous Spacecraft Guidance during Proximity Operations," *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics, 1 2021, 10.2514/6.2021-0668.

[14] F. E. Laipert and J. M. Longuski, "Automated Missed-Thrust Propellant Margin Analysis for Low-Thrust Trajectories," *Journal of Spacecraft and Rockets*, Vol. 52, 7 2015, pp. 1135–1143, 10.2514/1.A33264.

[15] D. Izzo, E. Öztürk, and M. Märtens, "Interplanetary transfers via deep representations of the optimal policy and/or of the value function," *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ACM, 7 2019, pp. 1971–1979, 10.1145/3319619.3326834.

[16] L. Cheng, Z. Wang, Y. Song, and F. Jiang, "Real-time optimal control for irregular asteroid landings using deep neural networks," *Acta Astronautica*, Vol. 170, 5 2020, pp. 66–79, 10.1016/j.actaastro.2019.11.039.

[17] D. Miller, J. A. Englander, and R. Linares, "Interplanetary low-thrust design using proximal policy optimization," *2019 AAS/AIAA Astrodynamics Specialist Conference*, No. GSFC-E-DAA-TN71225, 2019.

[18] B. Gaudet, R. Linares, and R. Furfaro, "Deep reinforcement learning for six degree-of-freedom planetary landing," *Advances in Space Research*, Vol. 65, 4 2020, pp. 1723–1741, 10.1016/j.asr.2019.12.030.

[19] A. Scorsoglio, R. Furfaro, R. Linares, and M. Massari, *Actor-Critic Reinforcement Learning Approach to Relative Motion Guidance in Near-Rectilinear Orbit*, Vol. 168. Advances in the Astronautical Sciences, 2 2019.

[20] C. J. Sullivan and N. Bosanac, "Using Reinforcement Learning to Design a Low-Thrust Approach into a Periodic Orbit in a Multi-Body System," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, 1 2020, 10.2514/6.2020-1914.

[21] N. B. LaFarge, D. Miller, K. C. Howell, and R. Linares, "Guidance for Closed-Loop Transfers using Reinforcement Learning with Application to Libration Point Orbits," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, 1 2020, 10.2514/6.2020-0458.

[22] J. Broida and R. Linares, *Spacecraft Rendezvous Guidance in Cluttered Environments via Reinforcement Learning*, Vol. 168. Advances in the Astronautical Sciences, 1 2019.

[23] A. Zavoli and L. Federici, "Reinforcement Learning for Robust Trajectory Design of Interplanetary Missions," *Journal of Guidance, Control, and Dynamics*, Vol. 44, 8 2021, pp. 1440–1453, 10.2514/1.G005794.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning : An introduction*. MIT Press, 2020.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," 2017.

[26] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," 2018.

[27] E. Noorani, C. N. Mavridis, and J. S. Baras, "Exponential TD Learning: A Risk-Sensitive Actor-Critic Reinforcement Learning Algorithm," *2023 American Control Conference (ACC)*, IEEE, 5 2023, pp. 4104–4109, 10.23919/ACC55779.2023.10156626.

[28] E. Noorani and J. S. Baras, "Embracing Risk in Reinforcement Learning: The Connection between Risk-Sensitive Exponential and Distributionally Robust Criteria," *2022 American Control Conference (ACC)*, IEEE, 6 2022, pp. 2703–2708, 10.23919/ACC53348.2022.9867841.

[29] E. Noorani and J. S. Baras, "A Probabilistic Perspective on Risk-sensitive Reinforcement Learning," *2022 American Control Conference (ACC)*, IEEE, 6 2022, pp. 2697–2702, 10.23919/ACC53348.2022.9867288.