AAS 20-689

USING MULTI-OBJECTIVE DEEP REINFORCEMENT LEARNING TO UNCOVER A PARETO FRONT IN MULTI-BODY TRAJECTORY DESIGN

Christopher J. Sullivan[*], **Natasha Bosanac**[†]

A multi-objective deep reinforcement learning algorithm, designated Multi-Reward Proximal Policy Optimization (MRPPO), is introduced to simultaneously train multiple policies, each with distinct reward functions. In this paper, MRPPO is used to uncover the Pareto front in a multi-objective optimization problem: designing a transfer for a low-thrust-enabled small satellite between two L_2 southern halo orbits in the Earth-Moon Circular Restricted Three-Body Problem (CR3BP). Once the policies are trained on this scenario, they are evaluated on a shared set of perturbed initial conditions to facilitate comparisons between policies and explore the time-of-flight and propellant mass usage trade space. A hyperparameter selection exploration is also performed to determine the influence of MRPPO's hyperparameters on the resulting behavior of the policies.

INTRODUCTION

Spacecraft conforming to the SmallSat form factor are of increasing interest to both scientists and mission designers for their potential to perform high-impact science for relatively low cost. The Mars Cube One (MarCO) mission recently demonstrated the capability for SmallSats to operate well beyond low Earth orbit (LEO), paving the way for additional missions such as Lunar IceCube, LunaH-Map, and NEA Scout.[1–3] Based on this precedent, an increasing number of mission concepts focus on the use of SmallSats to achieve innovative science and exploration objectives beyond LEO. During concept development, the SmallSat form factor and dependency on rideshare opportunities motivates simultaneous consideration of the spacecraft hardware, mission objectives, and trajectory geometry. In fact, due to limited thrust capabilities and a variety of operational constraints, a point solution for a trajectory that delivers the SmallSat from a fixed deployment condition to a desired destination is often sought to support feasibility assessments. Yet, simply designing and redesigning a feasible trajectory as the mission parameters evolve may be challenging and time-consuming. Furthermore, there are often multiple objectives that must be balanced during trajectory optimization, necessitating an exploration of the solution space. One approach to reducing the complexity of exploring the solutions to a multi-objective optimization problem is to study the Pareto front following the non-dominated solutions that balance two or more objectives. However, constructing a Pareto front requires recovering multiple feasible solutions that are significantly influenced by mission parameters. To enable rapid and efficient exploration of the solution space in multi-body trajectory design problems, particularly in applications where traditional trajectory design strategies may be challenging or time-consuming to apply, new methods must be developed.

[*]Ph.D. Student, Colorado Center for Astrodynamics Research, University of Colorado Boulder, Boulder CO, 80303.
[†]Assistant Professor, Colorado Center for Astrodynamics Research, Smead Department of Aerospace Engineering Sciences, University of Colorado Boulder, Boulder CO, 80303.

Deep Reinforcement Learning (DRL), which has seen increasing use within numerous industries, including the mission design community, offers one approach for rapidly recovering viable solutions.[4–10] Traditional DRL methods train neural networks to autonomously recover solutions that maximize the cumulative reward, defined using a single scalar reward function, over an entire trajectory.[11] As an extension, Multi-Objective Deep Reinforcement Learning (MODRL) methods simultaneously trains multiple policies, each with a distinct reward function formulation, by sharing environmental information across all policies. This approach enables a more efficient use of computational resources when compared with multiple, sequential runs of a traditional DRL method.

In this paper, a Multi-Reward Proximal Policy Optimization (MRPPO) algorithm is developed to extend the success of single reward Proximal Policy Optimization (PPO) to explore a multi-objective optimization problem. PPO has been used in several recent trajectory design investigations due to its favorable convergence properties in chaotic environments.[4–7, 12–14] MRPPO uses PPO as a building block, but augments the structure of the algorithm to enable parallel training of multiple policies and introduces information sharing by using the same propagation data gathered from the environment for all policies. Limiting the number of numerical integrations in an environment, which often commands a significant portion of the total training time, decreases the required computation time and resources during training. Sharing the propagation data also enables policies to learn from the innovative or disadvantageous actions undertaken by other policies, aiding the convergence of all policies. Additionally, policies with no agency in the environment may even be trained solely by learning from the actions taken by other policies. Since multiple policies, each with distinct objectives, are simultaneously trained on a single scenario, insights may be more efficiently generated into the design space. However, like PPO, the policies developed using MRPPO and the performance of the algorithm are influenced by hyperparameter selections. In this paper, the general procedure for implementing MRPPO and strategies for selecting suitable hyperparameters are outlined for multi-objective trajectory design in the Circular Restricted Three-Body Problem (CR3BP).

The MRPPO implementation is demonstrated in this paper using a low-thrust transfer for a SmallSat between two halo orbits at distinct energy levels near the Earth-Moon L_2 equilibrium point, a region of current interest.[15] Due to the difference in energy between the initial and final orbits, a heteroclinic connection between the two orbits does not exist. Thus, the SmallSat is modeled with a low-thrust engine; the additional continuous acceleration is used to adjust the energy and correct geometric differences to produce a low-thrust transfer. Due to the use of a low-thrust engine for maneuvering, transfer design often involves a balance between minimizing the time of flight and minimizing propellant mass usage. The resulting multi-objective optimization problem admits a Pareto front of optimal solutions that is recovered in this paper via MRPPO. By applying MRPPO to simultaneously train multiple policies with distinct reward function formulations and training parameters, the multi-objective trade space for this halo to halo transfer in the Earth-Moon system is explored. Once trained, the policies are evaluated on the same set of perturbed initial conditions and then used to study the relationship between the objectives across the set of nondominated solutions in this trajectory design scenario. Further, the total reward returned from three training runs for each set of distinct hyperparameter values is examined to gain insight into the hyperparameter selection process for a multi-body trajectory design scenario.

DYNAMICAL MODEL

In preliminary trajectory design activities within multi-body systems, the CR3BP is often used to approximate the motion of a spacecraft with negligible mass within the vicinity of two primary

bodies. In this section, the natural equations of motion for the CR3BP are presented, along with fundamental definitions necessary for characterizing and visualizing solutions. Then, these equations of motion are modified to incorporate the additional acceleration imparted by a low-thrust engine.

Circular Restricted Three-Body Problem

The motion of a spacecraft in cislunar space is modeled using the autonomous CR3BP. In this dynamical model, a body of negligible mass, e.g. a spacecraft, is assumed to be influenced by the gravity of two primary bodies, the Earth and Moon with masses M_1 and M_2 respectively, both described as point masses in circular orbits about their mutual barycenter. Then, a rotating frame, $(\hat{x}, \hat{y}, \hat{z})$, is constructed: the $\hat{x}$ axis follows the line-of-sight from the Earth to the Moon, the $\hat{z}$ axis is directed along the orbital angular momentum vectors of the two primaries, and the $\hat{y}$ axis is defined to create a right-handed coordinate frame. To facilitate comparisons between multi-body systems and aid in numerical integration, nondimensionalization of the position, velocity, time, and mass components is often employed via the distance, mass, and time characteristic quantities l^*, m^*, and t^*. A quantity that significantly influences the natural dynamical structures within a multi-body system is the mass ratio, defined as $\mu = M_2/(M_1 + M_2)$. Using these characteristic quantities, the state of a spacecraft is defined as $\bar{x} = [\bar{d}, \bar{v}]$ where $\bar{d} = [x, y, z]$ and $\bar{v} = [\dot{x}, \dot{y}, \dot{z}]$ are the position and velocity vectors of the spacecraft in the rotating frame with respect to the barycenter of the system. The equations of motion that govern the dynamics of the spacecraft in the CR3BP and in the rotating frame are written as

$$\ddot{x} - 2\dot{y} = \frac{\partial U^*}{\partial x} \quad \ddot{y} + 2\dot{x} = \frac{\partial U^*}{\partial y} \quad \ddot{z} = \frac{\partial U^*}{\partial z} \tag{1}$$

where the pseudo-potential function is $U^* = \frac{1}{2}(x^2 + y^2) + \frac{1-\mu}{d_1} + \frac{\mu}{d_2}$ and the spacecraft's distance to each primary is $d_1 = \sqrt{(x+\mu)^2 + y^2 + z^2}$ and $d_2 = \sqrt{(x-1+\mu)^2 + y^2 + z^2}$.[16] A constant of integration, denoted the Jacobi constant, $C_J = 2U^* - \dot{x}^2 - \dot{y}^2 - \dot{z}^2$, is an energy-type parameter that remains constant along a natural trajectory in the CR3BP. The Jacobi constant is valuable for describing the relative energy of states in the CR3BP; for a spacecraft to transfer between states with distinct Jacobi constants, a maneuver is required in the CR3BP.

The CR3BP admits a variety of natural dynamical structures that are useful in the trajectory design process. The five equilibrium points in the CR3BP, L_1-L_5, each admit periodic orbit families. One family of periodic orbits of significant interest is the Earth-Moon L_2 southern halo orbit family. The highly inclined members of this family possess a constant line-of-sight with the Earth, extensive coverage of the lunar surface, and are currently of interest to identify a mission orbit for the Lunar Gateway.[15] Figure 1 displays a subset of the Earth-Moon L_2 southern halo orbit family; each orbit is colored by the Jacobi constant. In this paper, two halo orbits with Jacobi constants of 3.11 and 3.07 are used for the initial and final orbits, respectively. Since these orbits possess distinctly different Jacobi constants, a low-thrust propulsion system is used to complete the associated transfer.

Low-Thrust-Enabled CR3BP

The additional acceleration supplied by a low-thrust engine and the propellant mass usage are incorporated into the equations of motion. First, the spacecraft is assumed to be equipped with three variable thrust, constant specific impulse, low-thrust engines aligned with the $\hat{x}, \hat{y}, \hat{z}$ axes of the rotating frame in the Earth-Moon system. Then, the acceleration imparted by the low-thrust

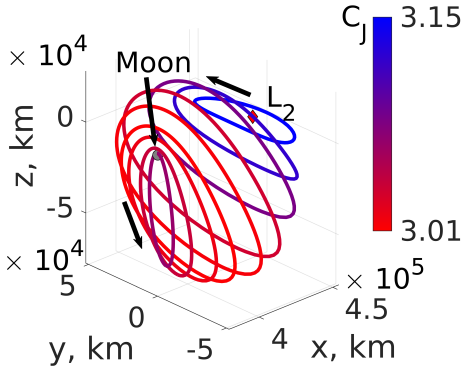


Figure 1: Members of the Earth-Moon L_2 southern halo orbit family shaded by Jacobi constant.

engine is

$$\bar{a} = \frac{T}{m_{s/c}} u_x \hat{x} + \frac{T}{m_{s/c}} u_y \hat{y} + \frac{T}{m_{s/c}} u_z \hat{z} = a_x \hat{x} + a_y \hat{y} + a_z \hat{z} \tag{2}$$

where T is the thrust magnitude for each low-thrust engine nondimensionalized by the spacecraft's initial wet mass, $m_{s/c}$ is the mass of the spacecraft nondimensionalized by the initial wet mass, and $\bar{u} = [u_x, u_y, u_z]$ is the control vector in the Earth-Moon rotating frame. The propellant mass usage is captured by the mass flow rate expression. When the low-thrust acceleration of the spacecraft is included in the equations of motion for the CR3BP, the complete system of equations is written as

$$\ddot{x} - 2\dot{y} = \frac{\partial U^*}{\partial x} + a_x \quad \ddot{y} + 2\dot{x} = \frac{\partial U^*}{\partial y} + a_y \quad \ddot{z} = \frac{\partial U^*}{\partial z} + a_z \quad \dot{m}_{s/c} = \frac{-T|\bar{u}|}{I_{sp} g_0} \tag{3}$$

where $|\bar{u}|$ is the magnitude of the thrust vector, I_{sp} is the specific impulse and $g_0 = 9.81$ m/s^2.[17] These seven differential equations define the dynamical model used within the DRL environment.

MULTI-OBJECTIVE DEEP REINFORCEMENT LEARNING

DRL has seen increasing use within the astrodynamics community for solving optimization problems in the CR3BP.[4–6] DRL algorithms are built using conventional RL as a foundation: an agent interacts with an environment by selecting actions with the objective of maximizing the long-term reward, as defined using a reward function.[18,19] The long-term reward is mathematically captured within the value function, evaluated for any state-action pair.[20] By gathering data in the environment and using that information within an update phase, a policy that maps actions to states is trained to maximize the value function for a defined environment and set of actions. DRL algorithms are distinct from RL methods due to their use of deep neural networks to approximate the policy and value function. Using the trained policy, locally optimal trajectories with respect to the value function may be calculated for any state within the training environment.

One category of algorithms that have seen increasing usage in DRL-type problems is actor-critic methods. These methods separate learning the policy from learning the value function by using two distinct neural networks, thereby aiding policy convergence and computational efficiency while training.[21] With this formulation, DRL algorithms train one neural network, the actor, to map the optimal action to every state while the other neural network, the critic, determines the states in the environment with the highest value. Numerous methods have been proposed to train the actor-critic

neural networks, but one that has demonstrated advantageous convergence properties in chaotic environments is PPO, which recovers a single policy per training session.[12–14] Given PPO's success in chaotic environments, a multi-reward architecture is constructed using PPO as a foundation that enables multiple policies to be trained in parallel, each with distinct reward function formulations.

Multi-objective RL algorithms are generally classified in three distinct categories: (1) a single policy learning multiple, time-dependent, scalar reward functions, (2) multiple policies learning vector-valued reward functions, and (3) multiple policies each learning a distinct scalar-valued reward function. MRPPO falls within the final category whereby multiple reward functions, where each may have a completely independent formulation, are assigned to an equal number of policies allowing each policy to learn behavior that maximizes the cumulative return from its designated reward function. These policies share their accumulated state-action propagation data, reducing the required computational resources and increasing the stability of the policies during training. Then, by simultaneously training multiple actor-critic deep neural networks using MRPPO, a Pareto front balancing two or more objectives within an environment may be generated.

Feed Forward Neural Networks

Neural networks offer one method for approximating highly nonlinear mappings between inputs and outputs via the construction of a web of nodes similar to the composition of neurons in a brain.[22] Feed forward neural networks accept the current state of the agent into the first layer of the neural network, denoted the input layer. Then, each node in the input layer is fully connected to each node in the proceeding layer, termed the first hidden layer. Each of these connections is assigned a weight that is altered throughout training via back propagation.[23] Then, the nodes in the first hidden layer take in each of the weighted inputs and feed them through an activation function, such as hyperbolic tangent (Tanh), rectified linear unit (ReLU), or sigmoid, to add nonlinearity to the function approximation.[24] For deep neural networks, this process is continued through multiple hidden layers, each with weighted connections $\bar{W}_i$, until the final hidden layer which is connected to the output layer, again using a set of weighted connections. The output may be represented using either a discrete set of actions, or more often for continuous action spaces, a probability distribution. Throughout the training process, deep neural networks offer a variety of advantages when compared to single hidden layer neural networks. First, deep neural networks are distinct from single hidden layer neural networks by the inclusion of two or more hidden layers in the neural network. There are various advantages for introducing additional hidden layers: (1) less nodes are required per layer to arrive at a local optimum, (2) training generally requires fewer iterations, and (3) deep neural networks tend to perform better in more complex scenarios than single hidden layer neural networks. Yet, one disadvantage of using deep neural networks is that they are generally not guaranteed to converge to the global optimum.[25] While constructing and evaluating feed forward neural networks requires a trivial amount of computational resources, training neural networks may be time-consuming and highly sensitive to the structure of the network.

Training a neural network centers upon updating the parameters of the network using experiences gathered from interacting with the environment. The parameters of the network include the weights between each layer and additional parameters that may be introduced to represent a probability distribution. For instance, for a Gaussian distribution for the actions, the output from the neural networks is generally used as the mean of the distribution, but additional parameters may be assigned to the network and trained for the standard deviation of the distribution. The training parameters for

the neural network, denoted $\bar{\theta}$ are then written as

$$\bar{\theta} = [\bar{W}_{in}, \bar{W}_{h,1,2}, \bar{W}_{h,2,3}, \ldots, \bar{W}_{out}, p] \tag{4}$$

where the weight vector between the input layer and first hidden layer is denoted $\bar{W}_{in}$, $\bar{W}_{h,1,2}$ is the weight vector for the connections between the first hidden layer and second hidden layer, $\bar{W}_{out}$ is the weight vector connecting the final hidden layer and the output layer, and p is a placeholder variable for any parameter not included in the neural networks that may be conditioned through training. For example, the standard deviation of a Gaussian distribution may be included as an additional parameter that is updated while training. Then, these parameters may be updated using a stochastic gradient descent optimization algorithm, which tend to perform well in the noisy, high-dimensional environments used in DRL. In this analysis, the Adam optimization algorithm is leveraged due to its efficient and well-tested convergence properties across many environments compared to other optimization algorithms.[14,26] Additionally, the number of hidden layers, number of nodes per hidden layer, and the activation function influence the training process.[14] For instance, more hidden layers and more nodes per hidden layer may allow the deep neural network to more accurately approximate the complex relationship between inputs and outputs at the expense of increased training times. Two deep neural networks with this structure are implemented within an actor-critic structure to uncover the solution space within a trajectory design scenario.

Actor-Critic Networks

State-of-the-art single objective DRL algorithms often leverage actor-critic methods to train a policy that maximizes the value function within an environment because of the advantages they offer over traditional policy learning methods. Namely, actor-critic methods divide learning the policy and value function into two independent processes: the actor and the critic. The actor learns the policy, which maps optimal actions to every state in the environment Meanwhile, the critic determines the states in the environment that maximize the cumulative reward, mathematically defined as the value function.[27] By constructing distinct structures to learn the policy and value function, the overall learning process is simplified, leading to robust convergence properties in complex environments. Further, the actor and critic are each constructed using deep neural networks to approximate the relationship between their inputs and outputs. The actor neural network approximates a policy function, denoted $\pi_\theta(\bar{u}_t|\bar{s}_t)$, which produces the optimal action, $\bar{u}_t$, for an input state, $\bar{s}_t$, at time t.[28] The inputs and outputs to the actor neural network may be vector or scalar quantities, but for this investigation, are defined as vectors. Conversely, the critic neural network learns to maximize the value function, $V^\pi(\bar{s}_t)$, written as

$$V^\pi(\bar{s}_t) = \sum_{t=0}^{\tau} \gamma^t r_t(\bar{s}_t, \bar{u}_t, \bar{s}_{t+1}) \tag{5}$$

where $r_t(\bar{s}_t, \bar{u}_t, \bar{s}_{t+1})$ is the reward function for the state and action at time t and the state at time $t+1$, γ represents the discount factor which discounts future rewards more heavily than current rewards, and τ denotes the maximum number of time steps for a trajectory in the environment.[29] An illustration of the structure of an actor-critic method is displayed in Fig. 2 where $\bar{\theta}_a$ and $\bar{\theta}_c$ are the parameter vectors of the actor and critic neural networks respectively. In this structure, both of the neural networks are initialized with no knowledge about the environment. However, by interacting with the environment via state-action-reward experiences, the neural networks may be trained to produce a locally optimal policy. Specifically, the environment encapsulates both the dynamical

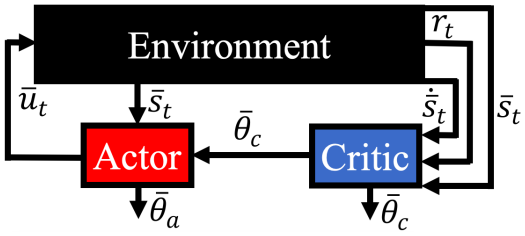


Figure 2: Actor-critic method interacting with an environment and sharing components between each structure.

model used to propagate states and actions and the reward function for each state action pair. While many methods exist to train the neural networks using these experiences, policy gradient methods offer good performance in complex, high-dimensional environments typical of trajectory design in multi-body systems.[13]

Policy Gradient Methods

Actor-critic methods often leverage policy gradient methods to train the neural networks due to their robust convergence properties in complex environments with continuous control.[13] Compared to other training methods, policy gradient methods enable training in continuous state, continuous action environments by gathering state-action-reward experiences and calculating the expected total reward for some performance estimator. The policy gradient, g', is written as

$$g' = \mathrm{E}\left[\sum_{t=0}^{\infty} A^{\pi}(\bar{s}_t, \bar{u}_t)(\nabla_{\bar{\theta}} \log(\pi_{\bar{\theta}}(\bar{u}_t|\bar{s}_t)))\right] \tag{6}$$

where $A^{\pi}(\bar{s}_t, \bar{u}_t)$ is the advantage function which evaluates how beneficial an action at a state is compared to the current policy's behavior.[30] However, the advantage function is usually unknown and must be estimated. The estimated advantage function, $\hat{A}_t$, may be evaluated using either a temporal difference method, which only considers the current time step and possesses high bias, or a Monte Carlo method, which evaluates the advantage using all future time steps but at the expense of introducing high variance. Generalized Advantage Estimator (GAE) combines both of these methods by weighting the current time step more heavily and discounting future time steps. Using GAE, the estimated advantage function is calculated via

$$\hat{A}_t^{\pi}(\bar{s}_t, \bar{u}_t) = \sum_{\ell=0}^{\infty} (\gamma\lambda)^{\ell} \delta_{t+\ell} \tag{7}$$

where

$$\delta_t = r_t(\bar{s}_t, \bar{u}_t, \bar{s}_{t+1}) + \gamma V^{\pi}(\bar{s}_{t+1}) - V^{\pi}(\bar{s}_t) \tag{8}$$

is the estimate advantage of $\bar{u}_t$ and λ is the GAE factor balancing the amount of bias and variance within the estimated advantage function.[30] Then, the policy gradient is rewritten to reflect the inclusion of the estimated advantage function via

$$\hat{g}' = \hat{\mathrm{E}}_t\left[\sum_{t=0}^{\infty} \hat{A}_t^{\pi}(\bar{s}_t, \bar{u}_t)(\nabla_{\bar{\theta}} \log(\pi_{\bar{\theta}}(\bar{u}_t|\bar{s}_t)))\right] \tag{9}$$

illustrating the estimated policy gradient, $\hat{g}'$, which is evaluated over a number of state-action-reward experiences.[12] However, the computation of the estimated policy gradient is nontrivial. Instead, an objective function is defined to ensure its gradient is equal to the estimated policy gradient. The policy gradient objective function, $L^{PG}(\bar{\theta})$, is written as

$$L^{PG}(\bar{\theta}) = \hat{\mathrm{E}}_t[\log(\pi_{\bar{\theta}}(\bar{u}_t|\bar{s}_t))\hat{A}_t^{\pi}(\bar{s}_t, \bar{u}_t)] \tag{10}$$

Then, $\hat{g}'$ is calculated by differentiating the objective function with respect to the policy's parameters.[12] However, this policy gradient function inhibits multiple policy gradient descent steps, a necessity for computationally efficient algorithms. Further, this loss function does not prevent large policy updates from destabilizing the networks, potentially leading to irreversible divergence.

To prevent rogue updates from destabilizing the neural networks, a trust region constraint may be employed to constrain the size of policy updates. Trust region policy optimization (TRPO) is one DRL algorithm that rewrites the objective function to impose a hard constraint on the size of the update.[29] The objective function and constraint for TRPO are defined as

$$\text{maximize}_{\bar{\theta}}: \hat{\mathrm{E}}_t\left[R_t(\bar{\theta}_j) \cdot \hat{A}_t^{\pi}\right] \tag{11}$$

$$\text{subject to: } \hat{\mathrm{E}}_t[KL[\pi_{\bar{\theta},j-1}(\cdot|\bar{s}_t), \pi_{\bar{\theta},j}(\cdot|\bar{s}_t)]] \leq \rho \tag{12}$$

where

$$R_t(\bar{\theta}_j) = \frac{\pi_{\bar{\theta},j}(\bar{u}_t|\bar{s}_t)}{\pi_{\bar{\theta},j-1}(\bar{u}_t|\bar{s}_t)} \tag{13}$$

is the probability ratio between the old and new policies at update j which approaches unity as the policy converges, KL refers to the Kullback-Leibler divergence, and ρ is the size of the trust region.[12,29] However, TRPO requires a nontrivial second order differentiation of the objective function to compute the estimated policy gradient, and the trust region to be defined a priori, limiting its effectiveness in certain applications.[12] To circumvent these limitations, PPO follows the structure of a policy gradient algorithm but incorporates characteristics of trust regions leading to advantageous performance over traditional policy gradient algorithms.

Proximal Policy Optimization

PPO was developed to simplify the practical implementation of TRPO while still incorporating the benefits of a trust region through a soft constraint in the objective function. Including a soft constraint in the objective function incentivizes policies to limit large changes, which is necessary in sensitive, chaotic environments, without loss of performance.[12] This soft constraint is implemented by a clipping parameter, ε, that scales any updates to the policy by bounding the probability ratio to within the range $(1 - \varepsilon \leq R_t(\bar{\theta}_j) \leq 1 + \varepsilon)$; this modification discourages the ratio from diverging away from unity.[12] Then, the objective function, L^{CLIP}, is written as

$$L_t^{CLIP}(\bar{\theta}_j) = \hat{\mathrm{E}}_t[\min(R_t(\bar{\theta}_j)\hat{A}_t, \mathrm{clip}(R_t(\bar{\theta}_j), 1 - \varepsilon, 1 + \varepsilon)\hat{A}_t)] \tag{14}$$

This objective function is augmented with two additional terms: (1) an entropy term that encourages exploration within the environment and (2) a squared-error loss term that encourages convergence towards an optimal policy by measuring the error of the estimated value function. This modified objective function that is used to compute the estimate policy gradient in PPO is written as

$$L_t^{CLIP+VF+S}(\bar{\theta}_j) = \hat{\mathrm{E}}_t[L_t^{CLIP}(\bar{\theta}_j) - c_1 L_t^{VF}(\bar{\theta}_j) + c_2 S[\pi_{\bar{\theta},j}](\bar{s}_t)] \tag{15}$$

where $L_t^{VF}(\bar{\theta}_j) = (V_\theta(\bar{x}_t) - V_t^{target})^2$ is the squared error loss term, $S[\pi_{\bar{\theta},j}](\bar{s}_t)$ denotes the entropy term which gradually decreases during training as the networks learn more about the environment, and c_1 and c_2 are scalar coefficients for the value error and entropy terms respectively.[12] Using this objective function enables both the actor and critic neural networks to be trained on an environment. Figure 3 illustrates a policy in blue, trained using PPO, controlling agents, depicted in red, that interact independent of one another in the environment, represented as a black box, to gather state-action-reward experiences. Then, after the agents gather state-action-reward experiences, the experiences are used to formulate an update to the policy denoted using the black summation symbol. This objective function formulation and structure provides relatively stable and efficient performance in complex environments compared to other state-of-the-art DRL algorithms for single objective problems.[13] PPO's proven performance in sensitive, chaotic environments supports using PPO as a foundation for a multi-reward, multi-objective framework that simultaneously trains multiple policies within a common dynamical model.

Multi-Reward Proximal Policy Optimization

MRPPO leverages the same objective function used by PPO, but expands the structure of PPO to enable multiple policies, each with distinct reward function formulations, to be trained simultaneously by sharing state-action propagation information across all policies. Since numerical integration often accounts for over 90% of the total training time for a policy, especially in a multi-body gravitational environment, sharing the propagation information with additional policies significantly reduces the required computational resources. Sharing propagation information also improves the performance and stability of each policy during training. In MRPPO, N policies, denoted π_i for $i = [1, \ldots, N]$, each control k_i agents, independently acting within the environment. Each policy receives the current state at time t of each assigned agent, and then uses the actor neural network to output actions for each agent to perform. Then, the dynamical model returns the state at time $t+1$ to generate a state-action transition, or $(\bar{s}_t, \bar{u}_t, \bar{s}_{t+1})$. This process is repeated until the maximum number, denoted τ, of state-action transitions have been generated for each policy. Then, the transitions for one policy are shared with every other policy; thus, each policy learns from an additional $(N-1)\tau$ transitions in the environment. Finally, each policy feeds the $N\tau$ transitions into its assigned reward function to drive its updates. This process is repeated until a terminating condition is reached; in this paper, termination is defined using a maximum number of updates. Figure 4 displays the structure of MRPPO for N policies in blue, each controlling k_i agents, depicted in red, the dynamical model is illustrated as a black box, the reward function for each policy is displayed in gold, and the black summation symbol represents where state-action pairs are stored prior to updating each policy.

While the training phase for MRPPO follows a similar configuration to PPO, the updates to the neural network parameters for MRPPO require modifications to the objective function. In MRPPO, actions selected by one policy must be evaluated for their value and log-likelihood by every other

Figure 3: Policy commanding agents in an environment and using the state-action-reward experiences to update the neural networks using PPO.

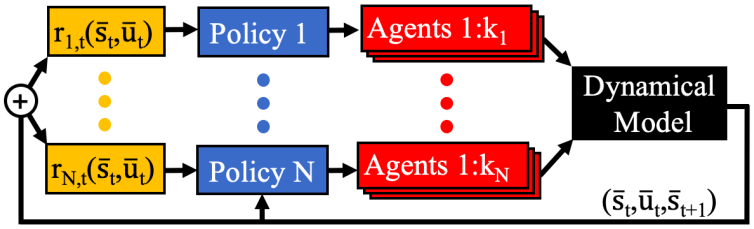


Figure 4: MRPPO architecture illustrating N policies, each with an assigned reward function $r_{i,t}(\bar{s}_t, \bar{u}_t)$, controlling k_i agents subject to the dynamical model.

policy during training. As the number of updates increases during training and policies begin to converge on their local optima, actions selected by one policy may become statistically unlikely for other policies, which is encapsulated within the log-likelihood values. However, the limitations imposed by machine precision cause some of these log-likelihood values to equal zero, thereby causing the denominator of the probability ratio defined in Eq. 13 to equal zero and causing the neural network parameters to immediately diverge. Instead, the probability ratio is redefined as

$$R_{i,t}(\bar{\theta}_{i,j}) = \pi_{\bar{\theta},i,j}(\bar{u}_t|\bar{s}_t) - \pi_{\bar{\theta},i,j-1}(\bar{u}_t|\bar{s}_t) \tag{16}$$

to inhibit statistically improbable actions from derailing the training process.[31,32] Then, once the policies begin to converge, the redefined probability ratio approaches zero rather than one. This necessitates that the clipping objective be rewritten as

$$L_{i,t}^{CLIP}(\bar{\theta}_{i,j}) = \hat{\mathrm{E}}_t[\min(R_{i,t}(\bar{\theta}_{i,j})\hat{A}_{i,t}, \mathrm{clip}(R_{i,t}(\bar{\theta}_{i,j}), -\varepsilon, \varepsilon)\hat{A}_{i,t})] \tag{17}$$

to include the new bounds, $[-\varepsilon, \varepsilon]$ on the probability ratio.[31,32] Then, MRPPO's structure enables a policy to experience potentially beneficial actions undertaken by other policies that the policy would not have performed itself, and discounts harmful actions, reinforcing default behavior in the event that a policy has converged on a near-globally optimal solution. With this formulation, MRPPO is used to train multiple policies with the same training parameters on a given scenario to produce locally optimal behavior for each policy and assigned reward function.

MRPPO TRAINING FACTORS

MRPPO is governed by multiple hyperparameters that influence the resulting behavior of the trained policies. While the hyperparameter selection process follows a high-dimensional multi-modal distribution where some hyperparameters are coupled, general insights may be generated by exploring the influence of each hyperparameter independently, relative to a baseline configuration. Hyperparameters that warrant further examination include:

- Environmental steps taken during training, τ: sets the number of state-action-reward experiences gathered using each agent prior to updating the parameters of the neural networks. Note that the total number of state-action-reward experiences used when updating the neural networks, Ψ, is a function of both the number of agents, K, and the number of environmental steps such that $\Psi = K \times \tau$. A larger number of steps tends to produce smoother convergence properties at the expense of longer training times and higher memory requirements.
- Epochs, E: control how many times those experiences are used to update the parameters.

- Mini batches, M: divides the total number of experiences into smaller sets of data, where each subset is then used to update the parameters. Figure 5 illustrates the relationship between epochs and mini batches for a single set of experiences demonstrating the total number of updates to the parameters of a neural network for one set of experiences is $E \times M$.

- Discount factor, γ: determines how heavily future rewards are discounted when computing the estimated advantages.

- GAE factor, λ: affects how future states are considered when calculating the advantages and reduces the variance in this computation.

- Learning rate, l_r: affects how quickly the neural networks are updated from a single set of state-action-reward experiences using the Adam optimizer.

- Clipping parameter, ε: limits the size of the updates to the networks in the objective function.

- Value function coefficient, c_1: scales the value loss error in the objective function.

- Entropy coefficient, c_2: determines the amount of entropy in the objective function.

Table 1 summarizes the hyperparameter values used as a baseline in this investigation, prior to hyperparameter exploration. These hyperparameters are selected due to their tendency to produce relatively stable performance during training across many trajectory design scenarios.

Table 1: MRPPO Baseline Hyperparameter Values.

Quantity	**Value**
Environmental Steps, τ	4096
Epochs, E	5
Mini Batches, M	4
Discount Factor, γ	0.99
GAE Factor, λ	0.99
Clipping Parameter, ε	2×10^{-3}
Value Function Coefficient, c_1	0.5
Entropy Coefficient, c_2	1×10^{-3}
Learning Rate, l_r	1×10^{-4}

A variety of construction parameters are defined that influence the resulting behavior of the policies during and after the training phase. For instance, the number of hidden layers and number of hidden nodes per layer for both the actor and the critic neural networks affect the ability of the policy to learn the optimal behavior. Further, the weight initialization scheme for the neural networks affects the trained policies due to the local basins of convergence that they may fall within; for instance, orthogonal, Xavier, and random techniques may each produce distinct results. Similarly, the activation function used within each layer influences the resulting trained policy. Finally, the action space representation, either by a Gaussian or beta distribution, may introduce bias into the training and affect the resulting training. The baseline values for these factors, identified through experimentation, are summarized in Table 2.

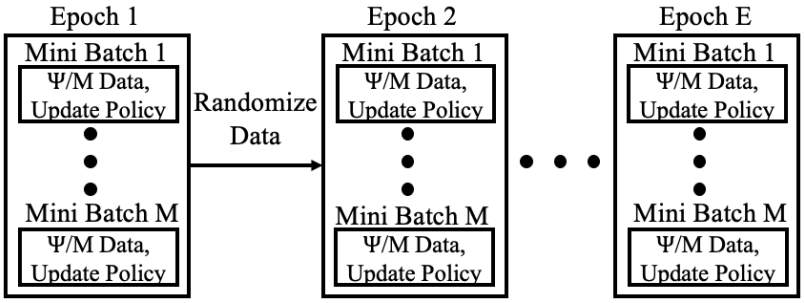


Figure 5: Epoch and mini batches used to update the neural networks for Ψ experiences where the data is randomized between each epoch to aid in convergence.

Table 2: MRPPO Baseline Construction Parameter Values.

Quantity	Value
Actor Hidden Layers	2
Actor Hidden Nodes per Layer	64
Critic Hidden Layers	2
Critic Hidden Nodes per Layer	1024
Weight Initialization Scheme	Orthogonal
Activation Function	Tanh
Action Space Representation	Gaussian

USING MRPPO TO RECOVER TRAJECTORIES IN THE CR3BP

MRPPO is used to train four policies on a transfer between two halo orbits for a low-thrust-enabled spacecraft in the CR3BP; the results are used to uncover portions of the Pareto front, demonstrating the propellant mass and flight time relationship. These results are also used as a mechanism for examining the impact of hyperparameter and construction parameters on the performance of the algorithm. This paper assumes a 180 kg ESPA-class SmallSat equipped with three low-thrust engines, each aligned with the $\hat{x}, \hat{y}, \hat{z}$ axes of the rotating frame. Each engine possesses variable thrust capabilities with a maximum available thrust of $T = 0.25N$ and a constant specific impulse of $I_{sp} = 3000s$. Then, the maximum total thrust available to the spacecraft at any time step is $T = 0.433N$. Using MRPPO, each policy must develop the mapping between optimal actions and states in the environment that maximizes the long-term reward from their assigned reward function. For this mapping, the state given to each policy consists of the spacecraft's state, mass, and difference between the spacecraft's state and closest orbit state in position space, written as

$$\bar{s}_t = [x, y, z, \dot{x}, \dot{y}, \dot{z}, m_{s/c}, \delta x, \delta y, \delta z, \delta \dot{x}, \delta \dot{y}, \delta \dot{z}] \tag{18}$$

where the final orbit is discretized into 10,000 states evenly spaced in time along the orbit. This enables the spacecraft to measure the difference between its current state and the closest state in position space along the final orbit to within a small error. Then, the actor neural networks are trained to map an optimal action to each state in the environment. The action vector is a 3×1 vector defined in the Earth-Moon rotating frame. If one of more components of the action vector has a magnitude greater than 1, the action component is scaled down to -1 or 1. Finally, the action vector is scaled by the thrust magnitude of each low-thrust engine. Once drawn, the action is held constant

in the rotating frame along a time step and the state is propagated forward in time in the low-thrust-enabled CR3BP for a time step of $\Delta t = 1 \times 10^{-2}$ nondimensional time units, or approximately 1 hour. This time step is selected to provide the neural networks with a reasonable amount of time to learn the results of their actions, but not too long where a non-ideal action selection may result in an irrecoverable trajectory. The policies are allowed a maximum of 314 steps to transfer from the initial halo orbit to the final halo orbit, which corresponds to a maximum transfer time of 3.14 nondimensional time units; slightly less than the orbital periods of the halo orbits. These policies are trained to guide a spacecraft from the initial orbit to the final orbit subject to a reward function.

In this scenario, the spacecraft is assumed to begin in the vicinity of a fixed point along the initial halo orbit and then transfer towards any point along the final halo orbit. The initial conditions on the initial halo orbit are centered on $\bar{x}_{IC} = [1.0855, 0, 0.0626, 0, 0.2735, 0]$, and the final orbit possesses initial conditions of $\bar{x}_F = [1.1484, 0, -0.1494, 0, -0.2192, 0]$. Then, perturbations are added to the initial conditions to reflect off-nominal conditions that may impact the controls scheme. These perturbations are drawn from a Gaussian distribution centered on the true initial condition with a standard deviation of 1×10^{-3} nondimensional units for both position and velocity components. Figure 6 demonstrates that without applied controls, these perturbed initial conditions rapidly depart from the vicinity of the initial halo orbit in less than one orbit period with many either leaving through the Earth-Moon L_2 gateway or impacting the Moon. While training, these initial conditions are randomly drawn after the spacecraft hits a termination condition. However, once the policies are trained, they are evaluated on an identical set of 25 perturbed initial conditions to facilitate comparisons between the trained policies. The evaluation initial conditions are separately selected from the training initial conditions to ensure that the policies have not experienced the exact initial conditions, to within machine precision, and also prevent overfitting which would bias the results. Then, the policies may converge on a locally optimal controls scheme that guides a spacecraft from perturbed initial conditions to the final orbit while maximizing the cumulative reward for each policy and reward function.

During training, the reward function formulation significantly influences the resulting behavior of the policies. To guide the spacecraft from the initial halo orbit to the final halo orbit, the reward function is structured to incentivize decreasing the position and velocity differences while conserving propellant mass. Then, the general reward function for this transfer is written as

$$r_t(\bar{s}_t, \bar{u}_t) = -10|\bar{d}_{ref} - \bar{d}_{s/c}(t+\Delta t)| - |\bar{v}_{ref} - \bar{v}_{s/c}(t+\Delta t)| - c_m(m_{s/c}(t) - m_{s/c}(t+\Delta t)) + \Omega \quad (19)$$

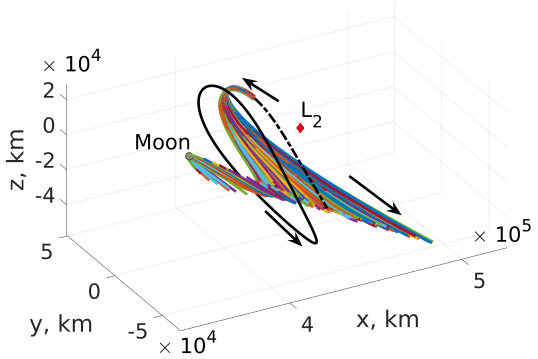


Figure 6: Natural trajectories emanating from perturbed initial conditions centered on a reference initial condition along a southern L_2 halo orbit in the Earth-Moon CR3BP.

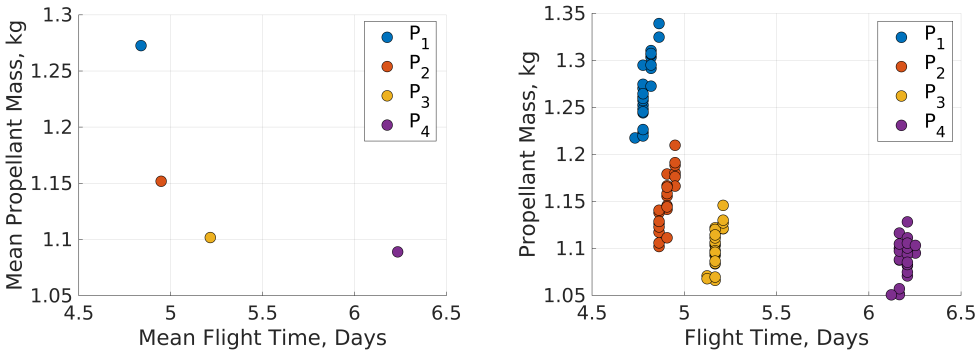
In this formulation, $\bar{d}_{ref}$ represents the position vector of the closest orbit state in position space to the spacecraft and the velocity vector of the closest state is $\bar{v}_{ref}$. The third term measures the propellant mass usage associated with the most recent action, $\bar{u}_t$, where c_m denotes the coefficient that scales the penalty on propellant mass usage. Finally, Ω is either a bonus or penalty term applied if one or more termination conditions is activated. Four termination conditions exist for this scenario: (1) the max number of time steps is reached and $\Omega = 0$, (2) the spacecraft approaches within 1×10^{-4} and 5×10^{-3} in position and velocity space, or 38.4 km and 5.1 m/s respectively, of the closest state in position space of the final halo orbit where $\Omega = 100$, (3) the spacecraft impacts the Moon where $\Omega = -1 \times 10^7$, and (4) the spacecraft departs through either the L_1 or L_2 gateways where $\Omega = -1 \times 10^7$. Note, the policies are only supplied information about the final orbit, without the use of a reference trajectory to guide the policies along a transfer between the orbits. In addition, each of the terms in the reward function is scaled to be approximately on the order of 10^0 throughout the transfer. With this formulation of the reward function, the variable mass coefficient creates a multi-objective trade space. By varying the coefficient on propellant mass usage, policies are trained to produce distinct behavior that maximizes the assigned reward function. Specifically, four policies are trained with mass coefficients of $c_m = [50, 60, 70, 80]$, corresponding to increasingly larger penalties for propellant mass usage along a low-thrust-enabled transfer for a SmallSat. The resulting policies and transfers are used to uncover the Pareto front associated with solutions that balance minimizing flight time and propellant mass usage.

RESULTS

Multiple policies, each with a distinct reward function, are trained to recover unique controls schemes for a transfer between two L_2 southern halos in the Earth-Moon system. Once trained, the four policies are evaluated on a common set of perturbed initial conditions drawn around the reference initial condition. Characteristics from the evaluation trajectories are used to develop the mean propellant mass usage and flight times for each controls scheme and a Pareto front is developed depicting the multi-objective solution space. Then, the effect of hyperparameter selection on the policies throughout training is evaluated using the total reward across all policies.

Developing a Single Pareto Front with MRPPO

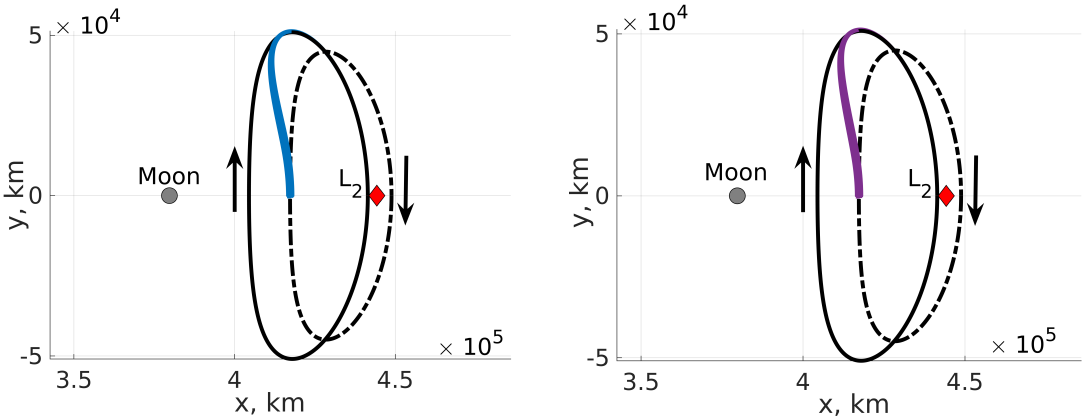
Four policies are simultaneously trained using MRPPO to develop controls schemes that guide a low-thrust-enabled SmallSat from perturbed initial conditions to an L_2 southern halo orbit in the Earth-Moon system using the baseline set of hyperparameters. Policy P_1 corresponds to the reward function with the lowest penalty on propellant mass usage with the proceeding policies corresponding to larger propellant mass penalties. Once the policies are trained and evaluated on the shared set of initial conditions, the mean flight time and propellant mass usage across each policy is calculated from the evaluation trajectories. Figure 7a depicts an estimate of the resulting Pareto front using the mean values from each uniquely colored policy. The Pareto front indicates, as expected, that a larger flight time corresponds to a lower required propellant mass for each policy. Additionally, Fig. 7b displays the flight time and propellant mass usages for all 25 evaluation trajectories for each policy, revealing consistent performance for each perturbed trajectory. Recall that each evaluation trajectory is perturbed differently; thus, slight differences in the trajectory characteristics are expected. The trajectories controlled by each policy to maximize distinct reward functions are also examined. For instance, Figs. 8a and 8b depict the trajectories evaluated for P_1 and P_4, shaded using the same color configuration as Fig. 7. Although the flight time and propellant mass across



(a) Pareto Front (b) Evaluation Trajectory Characteristics

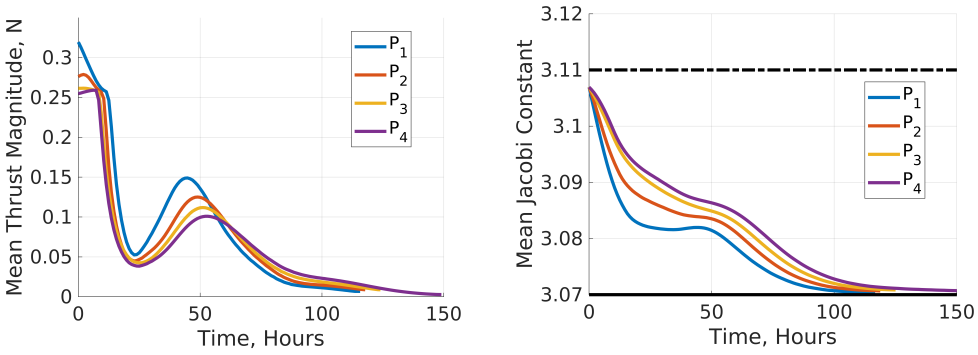
Figure 7: Characteristics of the evaluation trajectories for policies trained using MRPPO for: (a) mean characteristics to develop a subset of the Pareto front and (b) all trajectories' characteristics.

these solutions changes, there are marginal differences in geometry. This may be attributed to both policies reaching the vicinity of the final orbit after approximately the same number of time steps. However, once near the final orbit, P_1 tends to use more propellant mass to decrease the position and velocity differences faster while P_4 conserves propellant mass and more slowly approaches the position and velocity difference termination conditions. Figure 9a also depicts the average thrust magnitude for each policy as the penalty for propellant mass usage increases while Fig. 9b depicts the average Jacobi constant for each policy along with the Jacobi constants of the initial and final orbits as dashed and solid black lines, respectively. These two figures reveal that higher propellant mass usage penalties impact how quickly energy is added to the SmallSat. Specifically, P_1 increases the energy of the SmallSat faster than other policies while P_4 adds energy more gradually and consistently. Together, these results indicate the unique controls scheme recovered for policies that maximize distinct reward functions.



(a) Policy 1 (b) Policy 4

Figure 8: Perturbed trajectories controlled using: (a) Policy 1 and (b) Policy 4.



(a) Mean Thrust Magnitude

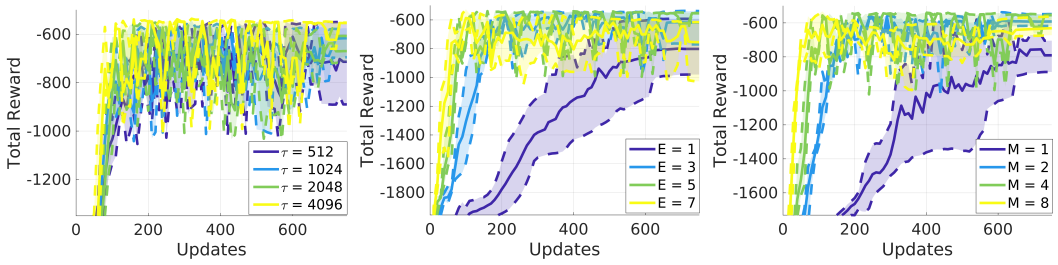
(b) Mean Jacobi Constant

Figure 9: Mean trajectory characteristics from the evaluation trajectories for all policies for: (a) thrust magnitude and (b) Jacobi constant.

Hyperparameter Exploration

Throughout training, the selected hyperparameters for MRPPO significantly influence the resulting policies: poor selections may prevent policies from converging to better locally optimal solutions. The trained policies for each set of hyperparameters are evaluated on the common set of perturbed initial conditions. Their effectiveness is evaluated using the total reward which is measured by summing the rewards from the evaluation trajectories for all four policies. The total reward is computed for every tenth update to the policies to study the evolution of the total reward throughout training. Additionally, three simulations of each set of hyperparameters are performed, and three values are used to determine their effectiveness: (1) the maximum total reward simulation, (2) the minimum total reward simulation, and (3) the mean of the total reward of all three simulations. These values are calculated to generate insight into how a hyperparameter affects the training of policies on this scenario using MRPPO. First, Fig. 10a depicts the average reward for distinct values of the number of environmental steps per update used during training. As the number of environmental steps used during training increases, the mean total reward slightly increases. This trend demonstrates the benefit of incorporating more information into the updates of the neural networks. The number of epochs and mini batches, however, displayed in Figs. 10b and 10c respectively, used during the update phase results in a more rapid increase in the total reward throughout training. Setting both hyperparameters to small values results in slower training and worse performance for all policies. However, high values also produce worse results. This observation may be attributed to both the sensitive environment and the data gathered in a single update being used to update the networks too many times, thereby biasing the training and forcing the policies to converge to sub-optimal results. Instead, a balanced approach is more suitable where the data is used to update the networks multiple times, but not too many so as to produce substandard policies.

Further insights are generated by examining the hyperparameters used within each update to the neural networks, including the discount factor, GAE factor, and learning rate, which influence how the data is used within a single update to the neural networks. Figure 11a depicts the influence of the discount factor, using three values. For this scenario, setting $\gamma = 0.95$ enables future rewards to be considered without under- or over-estimating their influence. Similarly, the GAE factor exploration displayed in Fig. 11b reveals that smaller values produce more consistent and improved performance for all policies, suggesting that too heavily accounting for future steps may harm the



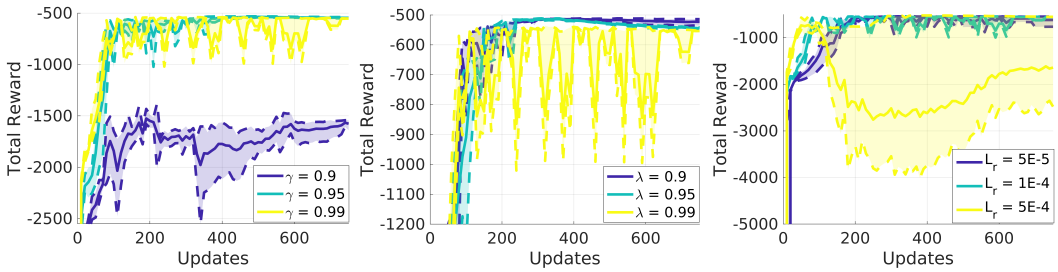
(a) Environmental Steps (b) Epochs (c) Mini Batches

Figure 10: Total reward for policies trained using MRPPO for various values of: (a) environmental steps, (b) epochs, (c) mini batches.

training process. While the values for the discount factor and GAE factor do influence the training, the value for the learning rate impacts training more significantly. In fact, poor learning rate choices result in policy divergence and suboptimal behavior. Figure 11c depicts the total reward for three learning rate values. The highest learning rate produces policies that significantly diverge while lower learning rates yield more consistent total rewards across policies. These three hyperparameter explorations demonstrate that properly selecting these values that govern the updates to the neural networks is a necessity for training policies with efficient and stable convergence properties.

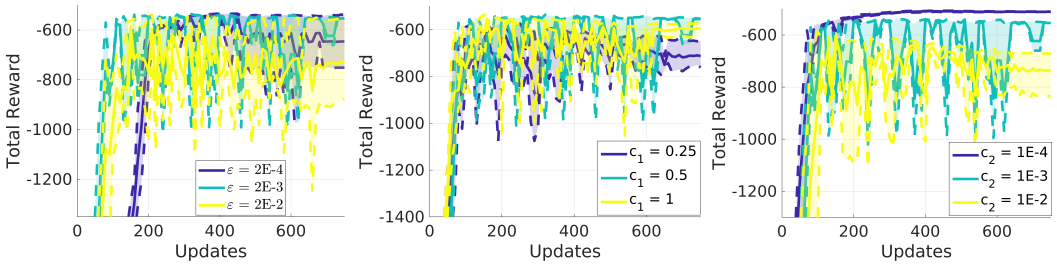
Three additional hyperparameters are used within the objective function defined in Eq. 15: the clipping parameter, value function coefficient, and entropy coefficient. Figure 12a depicts the results of three clipping parameter values illustrating that setting $\varepsilon = 2E - 3$ provides the most efficient convergence while also leading to the highest mean total reward. Then, Fig. 12b displays the influence of the value function coefficient with $c_1 = 0.5$ producing the highest maximum, minimum, and mean total rewards. Unlike the value function coefficient, Fig. 12c illustrates that the policies converge to higher total reward with a lower entropy coefficient in this scenario. Each of these hyperparameters directly affects the evaluation of the objective function during training leading to distinct behavior when evaluated on the same perturbed initial conditions.

Two construction parameters are also explored to analyze their influence on the trained policies. Figure 13a displays the results of using different initialization strategies for the weights of the neural



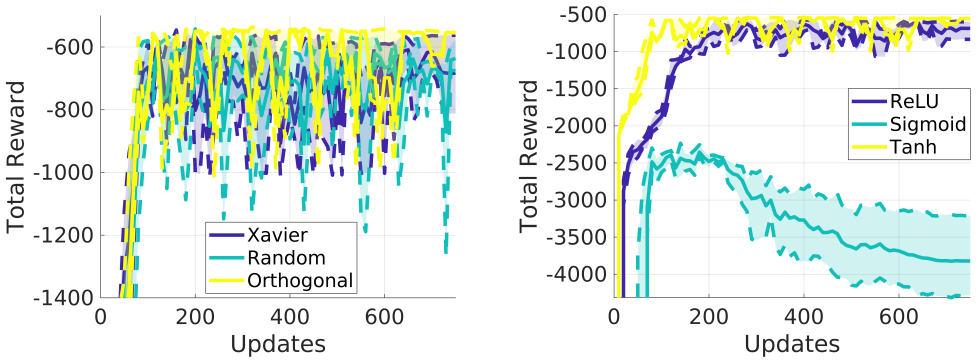
(a) Discount Factor (b) GAE Factor (c) Learning Rate

Figure 11: Total reward for policies trained using MRPPO for various values of: (a) discount factor, (b) GAE factor, (c) learning rate.



(a) Clipping Parameter (b) Value Function Coefficient (c) Entropy Coefficient

Figure 12: Average reward for policies trained using MRPPO for various values of: (a) clipping parameter, (b) value function coefficient, (c) entropy coefficient.



(a) Weight Initialization Method (b) Activation Function

Figure 13: Average reward for policies trained using MRPPO for various: (a) weight initialization methods and (b) activation functions.

networks. This figure reveals that the initial weights of each network slightly influence the resulting behavior of the policy in this scenario where an orthogonal initialization scheme produces higher maximum, minimum, and mean total rewards for all three simulations. Finally, Fig. 13b depicts the results of varying activation function used within the networks. Tanh converges towards the local maximum faster than using ReLU and provides more efficient and stable performance, which agrees with previous investigations on activation function performance in DRL.[14] In this scenario, these two construction parameters influence the resulting behavior of the policies trained using MRPPO.

CONCLUSION

An MODRL algorithm is designed using PPO as a foundation to enable multiple policies to be trained simultaneously in a shared environment with a multi-objective solution space. By sharing propagation data from the dynamical model, each policy may learn from the potentially beneficial or disadvantageous actions undertaken by other policies thereby introducing additional exploration into the training; this tends to produce a more stable, efficient training phase for MRPPO compared to PPO. Additionally, the structure of MRPPO reduces the computational resources and time required to train policies compared to sequentially training the policies using PPO. MRPPO is applied to a trajectory design scenario in the Earth-Moon system to develop four policies that maximize the

long-term reward from distinct reward functions. The policies are trained to guide a low-thrust-enabled SmallSat from a low-energy L_2 southern halo orbit to a higher-energy L_2 southern halo orbit within a single orbital period. This enables the recovery of a subset of solutions along the propellant mass usage and flight time Pareto front by calculating the mean flight time and propellant mass usage for each policy on a shared set of perturbed initial conditions. Once a Pareto front is developed for a set of policies with assigned reward functions, a preliminary investigation into the effect of hyperparameter selection is incorporated. For the selected hyperparameters and construction parameters used in MRPPO, a range of values is explored and compared to a default set of values to determine their influence on the resulting solutions. This analysis supplies preliminary example of selecting appropriate values for hyperparameters for applying MRPPO to trajectory design with multi-body systems.

ACKNOWLEDGEMENTS

This research was performed at the University of Colorado Boulder. This work was supported by a NASA Space Technology Research Fellowship. The authors thank Dr. Alinda Mashiku for serving as the research collaborator under this fellowship.

REFERENCES

[1] N. Bosanac, A. Cox, K. Howell, and D. C. Folta, "Trajectory Design for a Cislunar CubeSat Leveraging Dynamical Systems Techniques: The Lunar IceCube Mission," *Acta Astronautica*, 2018, pp. 283–296.

[2] A. L. Genova and D. W. Dunham, "Trajectory Design for the Lunar Polar Hydrogen Mapper Mission," *27th AAS/AIAA Space Flight Mechanics Meeting, San Antonio, TX*, 2017.

[3] L. Johnson, J. Castillo-Rogez, J. Dervan, and L. McNutt, "Near Earth Asteroid (NEA) Scout," *4th International Symposium on Solar Sailing, Kyoto Japan*, 2017.

[4] C. J. Sullivan and N. Bosanac, "Using Reinforcement Learning to Design a Low-Thrust Approach into a Periodic Orbit in a Multi-Body System," *30th AIAA/AAS Space Flight Mechanics Meeting, Orlando, FL*, 2020.

[5] D. Miller and R. Linares, "Low-Thrust Optimal Control via Reinforcement Learning," *29th AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI*, 2019.

[6] N. LaFarge, D. Miller, K. Howell, and R. Linares, "Guidance for Closed-Loop Transfers using Reinforcement Learning with Application to Libration Point Orbits," *30th AIAA/AAS Space Flight Mechanics Meeting, Orlando, FL*, 2020, 10.2514/6.2020-0458.

[7] A. Scorsoglio, R. Furfaro, R. Linares, and M. Massari, "Actor-Critic Reinforcement Learning Approach to Relative Motion Guidance in Near-Rectilinear Orbit," *29th AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI*, 2019.

[8] A. Das-Stuart, K. C. Howell, and D. C. Folta, "Rapid Trajectory Design in Complex Environments Enabled via Supervised and Reinforcement Learning Strategies," *69th International Astronautical Congress, Bremen, GER*, 2018.

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

[10] A. Harris, T. Teil, and H. Schaub, "Spacecraft Decision-Making Autonomy Using Deep Reinforcement Learning," *29th AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI*, 2019, pp. 1–19.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, 02 2015, pp. 529–33, 10.1038/nature14236.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," *Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA*, 2018.

[14] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study," 2020.

[15] D. C. Davis, S. M. Phillips, K. C. Howell, S. Vutukuri, and B. P. McCarthy, "Stationkeeping and Transfer Trajectory Design for Spacecraft in Cislunar Space," *2017 AAS/AIAA Astrodynamics Specialist Conference, Stevenson, WA*, 2017.

[16] V. Szebehely, *Theory of Orbits: The Restricted Problem of Three Bodies*. London, UK: Academic Press, 1967.

[17] D. A. Vallado, *Fundamentals of Astrodynamics and Applications*. Microcosm Press, 4th ed., 2013.

[18] M. Minsky, "Steps toward Artificial Intelligence," *in Proceedings of the IRE*, Vol. 49, No. 1, 1961, pp. 8–30. doi: 10.1109/JRPROC.1961.287775.

[19] A. G. Barto, "Reinforcement Learning and Dynamic Programming," *6th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design and Evaluation of Man-Machine Systems, Cambridge, MA*, Vol. 28, No. 15, 1995, pp. 407–412.

[20] J. A. Boyan and A. W. Moore, "Generalization in Reinforcement Learning: Safely Approximating the Value Function," *Advances in Neural Information Processing Systems 7* (G. Tesauro, D. S. Touretzky, and T. K. Leen, eds.), pp. 369–376, MIT Press, 1995.

[21] S. P. Singh, T. S. Jaakkola, and M. I. Jordan, "Learning Without State-Estimation in Partially Observable Markovian Decision Processes," *ICML*, 1994.

[22] K. Hornik, M. Stinchcombe, H. White, *et al.*, "Multilayer Feedforward Networks are Universal Approximators.," *Neural Networks*, Vol. 2, No. 5, 1989, pp. 359–366.

[23] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network," *Neural Networks for Perception*, pp. 65–93, Elsevier, 1992.

[24] C. E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," *CoRR, abs/1811.03378*, 2018.

[25] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, "Gradient Descent Finds Global Minima of Deep Neural Networks," *International Conference on Machine Learning*, 2019, pp. 1675–1685.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Algorithms," *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference, Denver, CO*, 2000, pp. 1008–1014.

[28] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable Trust-Region Method for Deep Reinforcement Learning Using Kronecker-Factored Approximation," *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 5279–5288, Curran Associates, Inc., 2017.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust Region Policy Optimization," *International conference on machine learning*, 2015, pp. 1889–1897.

[30] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[31] Z. Zhang, X. Luo, T. Liu, S. Xie, J. Wang, W. Wang, Y. Li, and Y. Peng, "Proximal Policy Optimization with Mixed Distributed Training," *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Portland, OR*, 2019, pp. 1452–1456.

[32] M. S. Holubar and M. A. Wiering, "Continuous-action Reinforcement Learning for Playing Racing Games: Comparing SPG to PPO," *arXiv preprint arXiv:2001.05270*, 2020.