(Preprint) AAS 19-779

INTERPLANETARY LOW-THRUST DESIGN USING PROXIMAL POLICY OPTIMIZATION

Daniel Miller [*], Jacob A. Englander[†], and Richard Linares[‡]

This paper aims to demonstrate a reinforcement learning technique for developing complex, decision-making policies capable of planning interplanetary transfers. Using Proximal Policy Optimization (PPO), a neural network agent is trained to produce a closed-loop controller capable of mass-optimal transfers between Earth and Mars. The agent is trained in an environment that utilizes a real ephemeris model of the Earth and Mars. Multiple scenarios are presented with both fixed and variable time steps. The results are compared against those generated by the Evolutionary Mission Trajectory Generator (EMTG) tool.

INTRODUCTION

The use of solar electric propulsion (SEP) in interplanetary missions has become increasingly important in recent years with its application on the BepiColombo, Dawn, Hayabusa, and Hayabusa2 missions. With an I_{sp} up to ten times greater than that of chemical propulsion, spacecraft utilizing SEP can reserve more payload for scientific equipment or cargo.[1]

Due to the low-thrust nature of SEP, burns are measured in several days or weeks, rather than the minutes and seconds of chemical propulsion. Determining the optimal sequence of thrust levels and directions over such a long, continuous period has proven challenging for traditional optimization techniques, as the curse of dimensionality can quickly take effect. Optimization methods can be broadly categorized as indirect or direct. Older, indirect methods numerically solve a two-bound boundary value problem, but their solutions may contain discontinuities in their control solution and are sensitive to initial conditions. Due to these limitations, direct methods have been extensively developed in the last twenty years. These approximate an optimal control problem as a nonlinear programming problem and were first applied to the design of low-thrust trajectories by Enright and Conway[2] and Sims and Flanagan.[3] Direct methods were later used to create software tools such as the Gravity-Assist Low-thrust Local Optimization Program (GALLOP)[4] and Mission Analysis Low Thrust Optimization (MALTO).[5] These optimizers utilize gradient-based methods that start from a user provided initial guess and, consequently, can fall victim to local minima in the region of the guess, rather than finding the true global minimum. Global search heuristics exist and may be paired with direct methods, but they can be expensive to operate.[6] Furthermore, direct methods

[*]Graduate Student, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02138. Email: dmmiller@mit.edu, Student Member AIAA.

[†]Aerospace Engineer, Navigation and Mission Design Branch, NASA Goddard Space Flight Center, Greenbelt, MD 20771. Email: jacob.a.englander@nasa.gov, Member AIAA

[‡]Charles Stark Draper Assistant Professor, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02138. Email: linaresr@mit.edu, Senior Member AIAA

are computationally expensive, which places limitations on the ability of the mission design to thoroughly search the solution space.

Another method for finding optimal solutions to low-thrust trajectory problems is Differential Dynamic Programming (DDP). Aziz, Scheeres, and Lantoine have used DDP to optimize low-thrust trajectories in the Earth-Moon circular restricted three-body problem[7] and trajectories for spacecraft propelled by SEP with power constraints due to solar eclipses.[8] Similar methods have also been used to optimize many-revolution transfers using DDP and a Sundman transformation.[9] As with the aforementioned direct methods, DDP is also computationally expensive and may require an initial guess.

Machine learning offers an alternative to these more conventional approaches to trajectory optimization. This broad field can be split into multiple categories including supervised learning and reinforcement learning (RL).[10] Supervised learning is the most familiar of these and is used for tasks such as image classification. In such a problem, an AI agent learns to pair training examples with their corresponding labels, all of which is provided by the researcher. In contrast, reinforcement learning involves an agent interacting with a researcher-designed environment to learn how to best maximize a reward signal. The use of multilayer Neural Networks (NNs) in machine learning, otherwise known as deep learning, is an area of intense study that has delivered advances in fields as divergent as image recognition and the board game Go.[11,12]

While the use of NNs in astrodynamics research is not necessarily new – Dachwald applied a shallow NN of only a single hidden layer to the low-thrust trajectory optimization problem in 2004[13] – it has become more prevalent in recent years. Recent applications include the determination of interplanetary trajectories with the assistance of genetic algorithms,[14] the identification of heteroclinic connections in the Circular Restricted Three Body Problem using NNs,[15] and planning a safe path through a cluster of satellites using the deep Q-learning RL algorithm.[16] Furfaro, Linares, and Gaudet explored the use of the RL algorithm Proximal Policy Optimization (PPO) for six degree-of-freedom Mars landing guidance,[17] while Broida and Linares used it to solve docking problems.[18] Furfaro et al. have also applied deep learning to determine fuel-optimal control solutions to lunar landing problems via simulated onboard lunar imaging.[19]

The research presented in this paper seeks to address the limitations of current low-thrust trajectory optimization methods through the use of the PPO reinforcement learning algorithm. First applied to low-thrust trajectory optimization by Miller and Linares,[20] PPO is a modern RL algorithm capable of training the complex policies necessary for neural network controllers. The trained agent will be used to solve a low-thrust Earth-Mars transfer and will be benchmarked against results provided by the Evolutionary Mission Trajectory Generator (EMTG), an interplanetary trajectory optimization tool provided by Goddard Space Flight Center. EMTG version 9 is a modular, scalable-fidelity trajectory optimization tool capable of both low fidelity global search and trade studies and high fidelity solution optimization.[6,21,22] The transfer used for comparison will be provided by EMTG's high-fidelity Parallel Shooting with Finite Burn (PSFB) optimization transcription.

Three training scenarios will be demonstrated. The first will be a 302 day transfer from the Earth to Mars split into 22 states separated by a consistent timestep. The second case will allow the agent to also control the integration time between each state, thus creating a variable time-step optimizer. Finally, the agent will also be trained on a variety of initial conditions representing launch conditions over a period of 2 weeks. This will gauge the ability of the agent to learn the underlying dynamics of the problem and its ability to generalize.

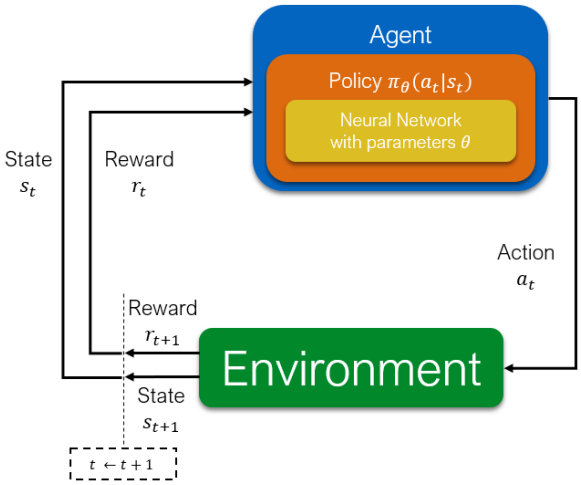


Figure 1: Simplified RL Process

REINFORCEMENT LEARNING

Preliminaries

Reinforcement learning is the training of an artificially intelligent agent to complete a task by allowing it to repeatedly attempt the problem and learn from feedback provided to it based on each decision it makes. This creates an elegant means of teaching an agent that does not require any previous knowledge of the problem. By freeing the training process of preexisting solutions, truly unique solutions may be found.[12]

The general process of RL is shown in Figure 1. An agent's actions are dictated by a policy function that provides the probability of action a_t given a state s_t. While some older RL algorithms utilize a table for this function, most methods now approximate it using a neural network composed of weights and biases, known collectively as the parameters, θ, of the NN. This is the $\pi_\theta(a_t|s_t)$ shown above. At a given time step, t, the agent will take an action a_t that is passed to the black-box problem environment. In the case of this research, this contains an orbital mechanics propagator and a reward calculator. From the environment, the agent then receives a reward based on its action and a state for the subsequent time step. This continues until some end conditions are met, such as leaving the bounds of the problem or exceeding a maximum number of time steps. Each sequence of states, actions, and rewards starting from an initial condition and terminating upon satisfying a predetermined end conditions composes a single episode of training. Having completed an episode, the agent can use this history of states, actions, and rewards to update the parameters of its policy to improve its performance.

Within RL, there is a subclass of algorithms known as actor-critic methods that utilize two neural networks: a policy network known as an actor and a second network called a critic that estimates the value of each state. Value, $V^\pi(s_t)$, is a measure of the quality of a given state and is an estimate of the future discounted rewards expected from the remaining states of the trajectory. A parameter γ is defined that discounts future rewards and controls how far into the future the agent will learn to consider when choosing an action.

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-t} r_T \tag{1}$$

Using values from the critic, the advantage of one action over another can be calculated. This is an estimate of the change in the value of a given state $V(s_t)$ as a result of the reward r_t provided in return for action a_t.

$$\hat{A}_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{2}$$

Policy Gradient and Trust Region Methods

Policy gradient methods are a large subcategory of RL algorithms that learn a policy function, $\pi(a_t|s_t)$, that yields the optimal action for a given state. In such a method, this policy is updated by using a gradient estimator, such as

$$\hat{g} = \hat{\mathbb{E}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{3}$$

This corresponds to the loss function

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{4}$$

According to Schulman, Wolsky, Dhariwal, Radford, and Klimov, repeatedly optimizing Eq. 4 using the same set of state, action, and reward tuples will often lead to destructively large policy updates.[23] This led to the creation of Trust Region Policy Optimization.[24] After collecting a trajectory of state, action, and reward tuples under the policy $\pi_{\theta_{old}}$, the parameters of a second policy π_θ is optimized using Eq. 5. To do so, the ratio of the probability of each action under the second policy is maximized with respect to the probability of the action as it occurred under the old policy, weighted by the advantages of each action, from Eq. 2. To counter the issue of large policy updates, a means of constraining the change in policy is required. By reason of the policy being a measure of the probability of an action given a state, the Kullback-Leibler divergence (KL) – a measure of the difference between two distributions – can be used to serve this purpose by limiting it to a value δ in Eq. 6.

$$\max_\theta \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \tag{5}$$

$$\text{subject to } \hat{\mathbb{E}}_t\left[\mathrm{KL}\left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right] \leq \delta \tag{6}$$

Proximal Policy Optimization

According to Schulman et al., while effective, TRPO is a complicated algorithm with limitations on its neural network architecture. In response to these issues, Proximal Policy Optimization (PPO) was designed to have the same state-of-the-art performance without the aforementioned disadvantages.[23] In order to achieve this, Eqs. 5 and 6 are replaced by a single loss function with a clipping factor instead of the KL-divergence constraint. Pseudocode for PPO can be found in Algorithms 1 and 2.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{7}$$

Algorithm 1 Proximal Policy Optimization (PPO) with continuous actions

Initialize actor neural network π_θ with parameters (weights and biases) θ
Initialize actor neural network $\pi_{\theta_{old}}$ with parameters θ_{old}
Initialize critic neural network V
Initialize LIFO $queue$ of length T
for episode = 1,2,... **do**
    while not Done **do**
        $a_t \sim \pi_{\theta_{old}}(a_t|s_t)$ ▷ Sample distribution for action
        $s_{t+1},\ r_t,\ Done \leftarrow Env(s_t, a_t)$ ▷ Take action a_t
        if $t \leq T$ **then**
            Record s_t and a_t in $queue$
        end if
    end while
    $targets \leftarrow TargetCalculation(s_{1:T}, a_{1:T})$
    $advantage_t \leftarrow target_t - V(s_t) \forall t$ ▷ Calculate Advantages
    Optimize L with respect to θ ▷ Update Actor
    Optimize Critic Network
    $\theta_{old} \leftarrow \theta$
end for

Algorithm 2 Target Calculation

procedure TARGETCALCULATION($s_{1:T}, a_{1:T}$)
    Initialize $TargetValues$ list
    $target \leftarrow 0$
    while $queue1$ not empty **do**
        $target_t \leftarrow r_t + \gamma * target_t$
        $TargetValues.append(target_t)$
    end while
    $TargetValues.reverse()$
    return $TargetValues$
end procedure

$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{8}$$

TRAJECTORY OBJECTIVE

To test the ability of the agent to learn a trajectory and its corresponding control solution, it was tasked with finding a 302-day fixed-length Earth to Mars transfer composed of 22 individual states. The first 21 of these states had corresponding actions and rewards that were used to train the agent. Since a reward is calculated based on the subsequent state that results from a given action, the final state would not be used. For initial conditions, a post-launch vector was provided by EMTG. The simulated spacecraft had an initial mass of $325.2\,\mathrm{kg}$ and used a $0.1\,\mathrm{N}$ thruster rated at an Isp of $3000\,\mathrm{s}$. The action space for both the throttle position and the thrust direction were completely continuous.

μ	L_{sc}	V_{sc}	T_{sc}
$1.327\,124\,401\,8 \times 10^{20}\,\mathrm{m^3\,s^{-2}}$	$149\,597\,870\,691\,\mathrm{m}$	$\sqrt{\frac{\mu}{L_{sc}}}$	$\frac{L_{sc}}{V_{sc}}$

Table 1: Dynamics Constants and Scaling Factors

DYNAMICS MODEL

The dynamics were modelled as a 2 body problem with continuous thrust and integrated forward using SciPy's *odeint* function. A sun-centered inertial frame was used to define the coordinate system and all units were nondimensionalized using the scaling factors shown in Table 1. Note that the length scaling factor, L_{sc}, is equivalent to 1 AU. The control input from the agent was provided in units of kg, since mass, unlike the length and time scaling factors, is a function of time and the control trajectory. The thrust was therefore nondimensionalized during the integration step in the equations of motion. These are shown in Equation 9, while the mass flow rate is shown in Equation 10.

$$\begin{aligned} \ddot{x} &= -\frac{x}{r^3} + \frac{u_x}{525.2\,\mathrm{kg}\cdot m} \\ \ddot{y} &= -\frac{y}{r^3} + \frac{u_y}{525.2\,\mathrm{kg}\cdot m} \\ \ddot{z} &= -\frac{z}{r^3} + \frac{u_z}{525.2\,\mathrm{kg}\cdot m} \end{aligned} \tag{9}$$

$$\dot{m} = -\frac{\sqrt{u_x^2 + u_y^2 + u_x^2}}{525.2\,\mathrm{kg}\cdot m} \cdot \frac{9.806\,65\,\mathrm{m\,s^{-2}}\cdot T_{sc}^2}{L_{sc}} \cdot \frac{3000\,\mathrm{s}}{T_{sc}} \tag{10}$$

IMPLEMENTATION

In any RL problem, the observation variables – the input to both the policy and value NNs – and the reward function are of critical importance. The former needs to fully encapsulate all of the information that the agent requires in order to make a decision. For the observation, the agent was provided with the position and velocity of the agent, its position and velocity error with respect to Mars, its mass, and current time step. In cases two and three, the current mission time was also provided.

$$s_t = \{x, y, z, \dot{x}, \dot{y}, \dot{z}, m, x_{err}, y_{err}, z_{err}, \dot{x}_{err}, \dot{y}_{err}, \dot{z}_{err}, t, t_{step}\} \tag{11}$$

All observation variables were nondimensionalized so as to have similarly scaled maximum values. Distances, velocity, and time were nondimensionalized using the scaling factors found in Table 1. The initial mass was scaled to a value of 1. The final two observation variables – mission elapsed time and the current time step – were also scaled such that their maximum values achieved at the end of the trajectory would also be 1.

An agent will always try to learn how to best exploit the feedback signal provided by its environment. As a result, this signal, the reward function, must be carefully designed; if it does not accurately reward or punish actions, a suboptimal policy will surely result. This complicates the implementation of hard constraints, such as a required flight time. Due to the use of a single objective that must contain both the function to be optimized and any constraints, it is possible that the agent

Case	c_1	c_2	c_3	c_4	c_5	c_6
Case 1	0.5	-500	-0.05	-500	0.25	0
Cases 2	0.5	-250	-0.05	-250	0.025	0.25
Cases 3	0.5	-500	-0.05	-500	0.025	0.1

Table 2: Reward Function Coefficients

may enforce a constraint at the expense of the task itself. As a result, all of the terms in the reward must be carefully balanced to ensure that the agent optimizes for all of the desired objectives.

A general framework of linear and exponential terms was formed with experimentally determined coefficients to weight each component. These coefficients are shown in Table 2. The first two terms are negative linear terms which encourage the agent to minimize the position and velocity error with respect to Mars. The third and fourth terms are positive exponential terms that provide a steep positive signal for minimizing the position and velocity errors when the error is small. This ensures that the fine policy improvements late in training have a clear reward. The fifth term is the fuel optimality penalty and tasks the agent with minimizing the change in spacecraft mass. The sixth term is to ensure that the agent meets the 302-day fixed time requirement for the mission and is a function of elapsed mission time measured in days. It is only activated when calculated the reward for the final state-action pair of the trajectory, since a time penalty for undershooting 302 days cannot be calculated until the final time is determined. The seventh and final term, p_t, is used as an additional penalty to constrain the region of space in which the agent can explore. If any are activated, the episode immediately ends by activating the $Done$ boolean in PPO Algorithm 1 and the agent is penalized. Please note that multiple penalties could be incurred simultaneously.

$$\begin{aligned} r_t = &- \left\| s_{t_{pos.\,error}} \right\| - \left\| s_{t_{vel.\,error}} \right\| + c_1\, e^{c_2 \left\| s_{t_{pos.\,error}} \right\|} + c_3\, e^{\left\| c_4 s_{t_{vel.\,error}} \right\|} \\ &- c_5(1-m) - c_6 p_{time_t} + p_t \end{aligned} \tag{12}$$

$$p_{time_t} = \begin{cases} |302 - t_{days}| & \text{if } \|x, y, z\| < 0.25 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

$$p_t = \begin{cases} -20 \text{ \& done} & \text{if } \|x, y, z\| < 0.25 \\ -20 \text{ \& done} & \text{if } \|x, y, z\| > 1.75 \\ -20 \text{ \& done} & \text{if } \|z\| > 0.75 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

A simple neural network architecture of 3 fully connected layers was chosen for both the policy and value NNs. The network size and activation functions can be found in Table 3. The architecture for Cases 1 and 2 were based on the previous work of Miller and Linares.[20] For Case 3, a larger network based on the work of Gaudet, Linares, and Furfaro[17] was found to be more effective. The optimization step was completed using Adam optimizer[25] using the learning rates found in Table 4. The learning rate is another name for the step size of a gradient descent optimizer. These values were kept high early in training to allow for larger upates to the NNs and then decreased later for finer tuning. The code was written in Python 3.7 using the popular Tensorboard-GPU library and run on a Lenovo ThinkPad P51 with an Intel Xeon E3-1505M V6 CPU and an Nvidia Quadro M2200 GPU.

		Cases 1 & 2		Case 3	
		Width	Activation	Width	Activation
Layer 1	Policy	32	ReLU	150	ReLU
	Value				
Layer 2	Policy	32	ReLU	87	ReLU
	Value			27	
Layer 3	Policy	32	ReLU	50	ReLU
	Value			5	
Output	Policy	4	tanh	5	tanh
	Value	1	linear	1	linear

Table 3: Neural Network Architectures

	Case 1	
Episodes (thousands)	Actor LR	Critic LR
0-25	0.0001	0.0005
25-45	0.00005	0.00025
45-60	0.000025	0.000125
60-100	0.00001	0.00005

	Case 2	
Episodes (thousands)	Actor LR	Critic LR
0-25	0.0001	0.0005
25-35	0.00005	0.00025
35-45	0.000025	0.000125
45-75	0.00001	0.00005
75-125	0.000005	0.000025

	Case 3	
Episodes (thousands)	Actor LR	Critic LR
0-50	0.0001	0.0005
50-100	0.00005	0.00025
100-125	0.000025	0.000125
125-150	0.00001	0.00005

Table 4: Adam Optimizer Learning Rate Schedule

	Dimensionalized	Non-dimensionalized
x	$130\,325\,632.242\,480\,67\,\mathrm{km}$	0.8711730430419905
y	$66\,016\,869.369\,941\,33\,\mathrm{km}$	0.4412955148693369
z	$28\,620\,014.942\,336\,10\,\mathrm{km}$	0.19131298333418004
$\dot{x}$	$-11.891\,569\,20\,\mathrm{km\,s^{-1}}$	-0.3992510403276126
$\dot{y}$	$27.870\,097\,74\,\mathrm{km\,s^{-1}}$	0.9357188550630681
$\dot{z}$	$13.484\,934\,82\,\mathrm{km\,s^{-1}}$	0.45274716608763854
m_0	$525.2\,\mathrm{kg}$	1
t	$2\,454\,396.306\,623\,12\,\mathrm{days}$	0
t_{step}	0	0

Table 5: Initial Conditions

RESULTS

Case 1: Single Initial Condition, Fixed Time Step

As an initial baseline demonstration, the agent was trained from a single initial condition (Table 5) on a trajectory composed of 22 states with equal time spacing between them of approximately 14.38 days each. Any action taken at an individual state would be held constant as the spacecraft was propagated over the length of the timestep. The post-launch initial condition was provided by EMTG and is used for a comparison of the delivered mass to Mars.

Table 6 shows the error between the Mars and the spacecraft's final state. The agent was able to achieve errors on the order of at least 10^{-3}, however, due to the scaling length of 1 AU, this is still substantial value. It is interesting to note that as a result of the scaling, the position and velocity had errors of the same order of magnitude, even though their dimensionalized values are some seven orders of magnitude apart. This suggests that the accuracy achieved by the agent is related to the learning process itself – the NNs, the reward function, and the PPO algorithm – rather than the physics of the environment. Shown in Fig. 4, the agent delivered $479.4616\,\mathrm{kg}$ to Mars, compared to the optimal final mass of $481.0995\,\mathrm{kg}$. In nondimensionalized units, this is an error of $3.1186e-3$, which is consistent with the order of magnitude of the total position and velocity errors.

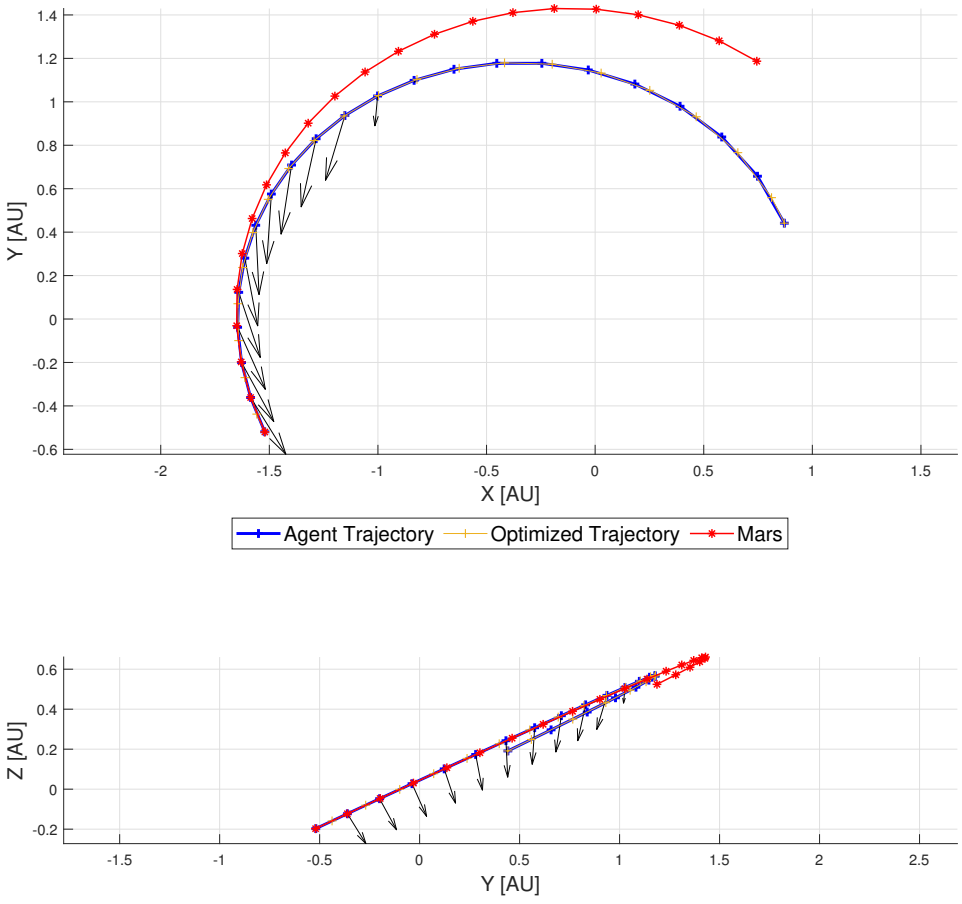


Figure 2: Case 1 Trajectory

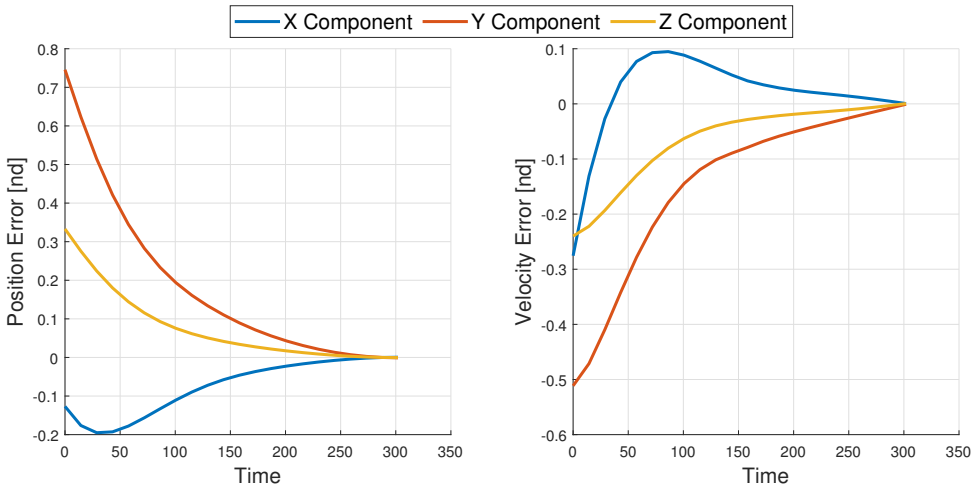


Figure 3: Case 1 Position and Velocity Error

		Position Error			Velocity Error		
Components	[nd]	5.7019e-04	-1.0363e-03	-2.7277e-04	5.3032e-04	-7.6738e-04	4.9067e-04
	[km]	8.5299e+04	-1.5503e+05	-4.0806e+04	1.5795e-02	-2.2856e-02	1.4614e-02
Total	[nd]	1.2138e-03			1.0540e-03		
	[km/s]	1.8159e+05			3.1392e-02		

Table 6: Case 1 Final State Error

Case 2: Single Initial Condition, Variable Time Step

Case 2 introduced a new variable for the agent to control: time step length. Along with a thrust magnitude and direction, the agent could now also select a length of time between 5% and 100% of the maximum allowable value of $2\,544\,047.999\,997\,586\,\mathrm{s}$, or approximately $29.4\,\mathrm{days}$. The total mission time was still constrained to $302\,\mathrm{days}$ by the 6th term in equation 12 and the trajectory was still composed of 22 states. The purpose of this test was to determine if the agent could intelligently time its actions and, if so, whether this would improve its performance.

As shown in Fig. 5, the agent chose to spread out its actions during the coast phase at the start of the transfer and to place them increasingly close together as it approached Mars. This was the expected behavior, for it allowed the agent to change its actions at more advantageous points in the state space. An alternative explanation is that the agent was also taking advantage of the rewards being higher at the end of the transfer when it would be in close proximity to Mars.

The constraint on elapsed time proved effective, with the agent undershooting the desired 302 day transfer by approximately $1\,\mathrm{h}$ and $21\,\mathrm{min}$. Regarding the final error, the use of variable time steps had a minimal impact, with the final errors in Table 7 being similar to those of Case 1. At $1.2386\,\mathrm{kg}$ above the EMTG value, the delivered mass error was likewise close to that of the fixed time step trajectory.

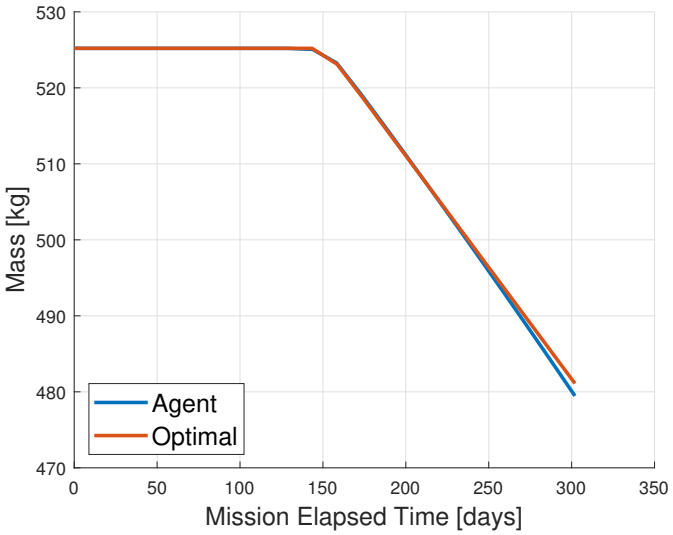


Figure 4: Case 1 Spacecraft Mass

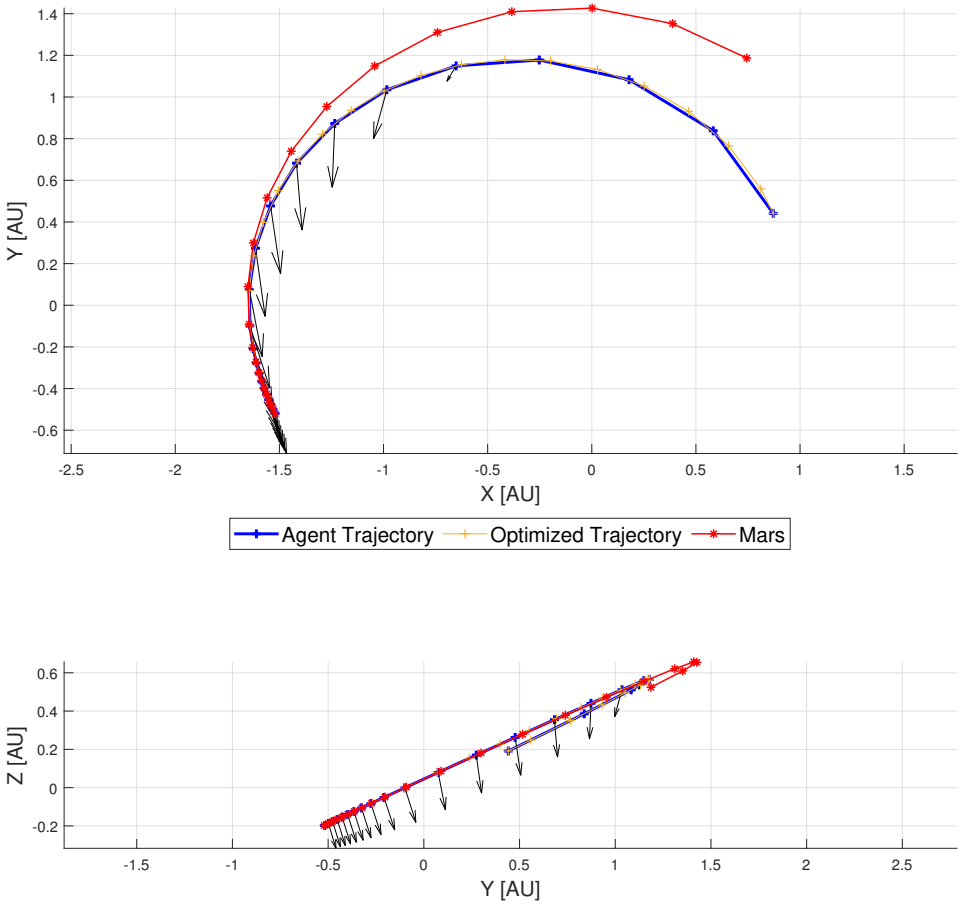


Figure 5: Case 2 Trajectory

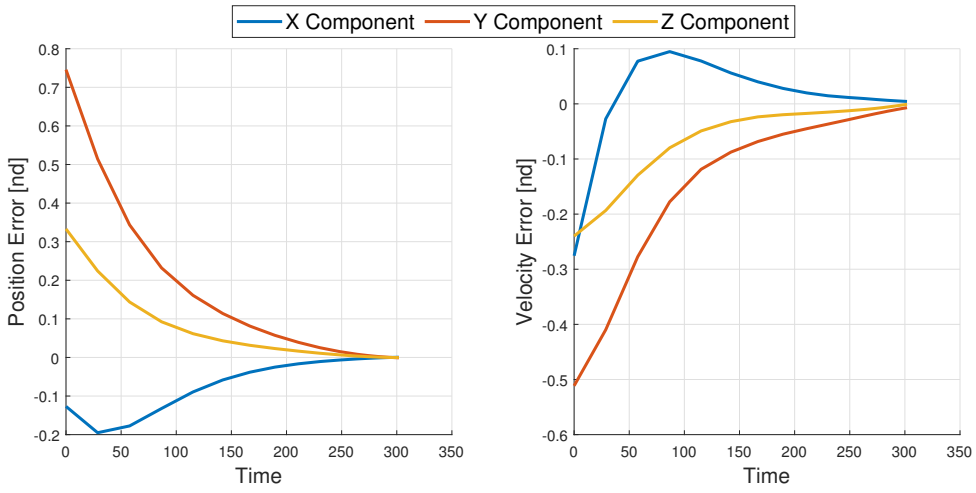


Figure 6: Case 2 Position and Velocity Error

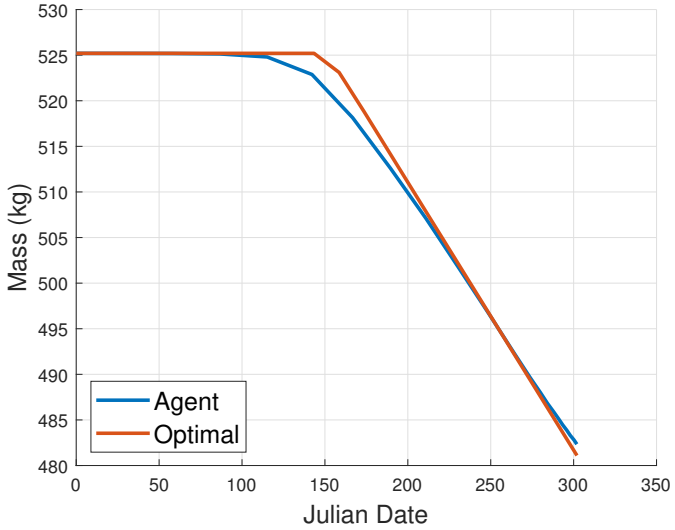


Figure 7: Case 2 Spacecraft Mass

		Position Error			Velocity Error		
Components	[nd]	6.0574e-04	-9.4371e-04	-2.3980e-04	4.3802e-03	-6.9200-03	-7.7607e-04
	[km]	9.0617e+04	-1.4118e+05	3.5874e+04	1.3046e-1	-2.0610e-1	-2.3115e-02
Total	[nd]	1.1467e-03			8.2263e-03		
	[km/s]	1.7155e+05			2.4502e-01		

Table 7: Case 2 Final State Error

Julian Launch Date [Days]	x [km]	y [km]	z [km]	$\dot{x}$ [km/s]	$\dot{y}$ [km/s]	$\dot{z}$ [km/s]
2454390.5	137126535.2	53754294.13	23304277.85	-9.50459523	28.63515046	13.87306768
2454391.5	136051906.9	55908732.72	24238248.17	-9.92156424	28.51356321	13.81377522
2454392.5	134936629	58046250.44	25164863.76	-10.33609713	28.38747457	13.75173512
2454393.5	133781050.6	60166198.71	26083846.18	-10.74837088	28.25804376	13.68503293
2454394.5	132585533.5	62267942.93	26994924.01	-11.1581895	28.12369961	13.616511
2454395.5	131350450	64350863.89	27897833.52	-11.56531132	27.985202	13.54464304
2454396.315	130314470.6	66034720.44	28627753.04	-11.89525083	27.86877604	13.48448934
2454396.5	130076172.4	66414370.03	28792323.91	-11.96980794	27.84203116	13.47055617
2454397.5	128763103.3	68457841.11	29678132.67	-12.37158976	27.6944174	13.3937673
2454398.5	127411589.6	70480754.6	30555039.94	-12.77036637	27.54219196	13.31482943
2454399.5	126022041.3	72482491.02	31422783.47	-13.16622655	27.38554858	13.23341032
2454400.5	124594788.5	74462539.62	32281147.08	-13.55746085	27.22465757	13.14950161
2454401.5	123130195	76420324.5	33129884.58	-13.94579305	27.05945782	13.06342681
2454402.5	121628611.3	78355276.58	33968750.63	-14.32995975	26.8899035	12.97538455
2454403.5	120090392.073	80266814.9579	34797492.5776	-14.71027669	26.71603922	12.88529796

Table 8: Case 3 Initial Conditions

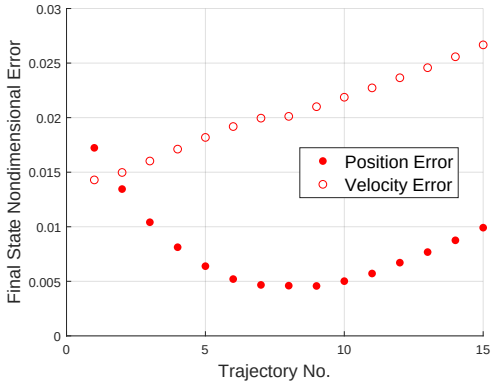
Case 3: Multiple Initial Conditions

In Case 3, the ability of the agent to generalize and complete the transfer from a range of initial conditions was analyzed. Each Earth-Mars transfer would still be composed of 22 states over an intended 302 day transfer, but the initial state would by randomly selected from a set of 15 post-launch vectors provided by EMTG spread over 13 days. See Table 8 for the list of initial conditions.

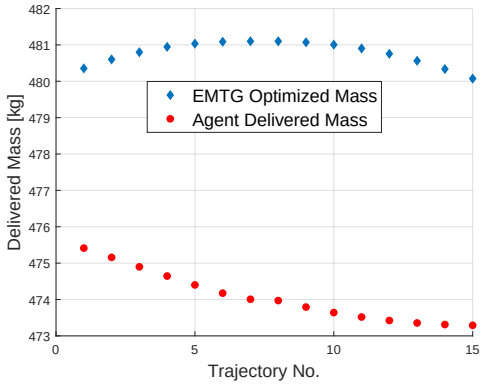
The results are shown in Figure 8. In Subfigure (a), the final position and velocity error is shown. It can be seen that the position error varies between $4.5755e-3$ and $1.7233e-2$, while the velocity error varies from $1.4293e-2$ and $2.6669e-2$. This is a degradation of 1-2 orders of magnitude from the results of Case 2. Subfigure (b) shows that the agent delivered between $4.9410\,\mathrm{kg}$ and $7.3841\,\mathrm{kg}$ less mass to Mars, as compared to the corresponding EMTG solutions. In Subfigure (c), it can be seen that the agent struggled with meeting the 302 day mission time constraint. Finally, Subfigure (d) shows the sum of rewards over each transfer and provides a measurement of how successful each transfer was. The greater the reward, the closer it came to optimizing the position, time, and mass requirements. Interestingly, while Subfigures (a), (b), and (c) each had its own trend lines, they broadly did not match that of (d). The exception to this is the position error of (a). Based on these similar trends, it can be concluded that the position error in the third term of Eq. 12 largely dominated the performance of the agent. Further experimentation with the reward function coefficients would be beneficial for better balancing the position, velocity, mass, and time performance of the agent across all of the trajectories.

CONCLUSION

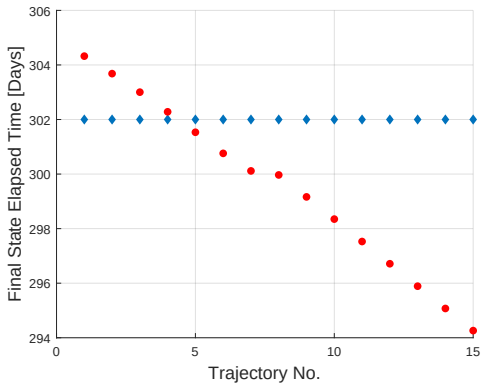
This paper demonstrates the application of the Proximal Policy Optimizer RL algorithm to a mass-optimal low-thrust transfer between the Earth and Mars. Without any prior knowledge of the problem and only being provided with information regarding its position and velocity relative to Mars, the agent was successfully able to learn a closed-loop guidance policy that could accomplish this task. When trained on a single trajectory, the agent was able to achieve position and velocity errors on the order of $10^5\,\mathrm{km}$ and $10^{-1}\,\mathrm{km\,s^{-1}}$, respectively. Compared to an optimal trajectory provided by EMTG, the delivered mass error was under $2\,\mathrm{kg}$. The agent also demonstrated the ability to vary the integration time between states in order to apply higher fidelity control where it was required. When trained on a range of initial conditions corresponding to a 15 day period of



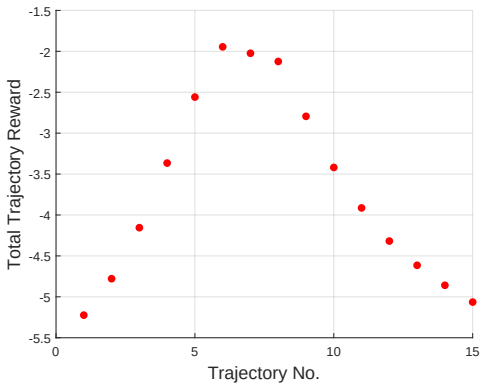
(a) Final State Error with respect to Mars



(b) Delivered Mass



(c) Final Elapsed Time



(d) Sum of Trajectory Rewards

Figure 8: Case 3 Results

launch dates, the agent demonstrated that it could still complete the transfer, although with some degraded performance.

In the future, further research will be required in both reducing the final errors and in improving the ability of the agent to generalize across numerous trajectories. Based on the similar orders of magnitude of the various nondimensionalized errors in Cases 1 and 2, the large final errors may relate to the scaling of the environment or could be a limitation of the either the algorithm or the neural network architecture. To improve the agents ability to generalize, the use of batching multiple trajectories may be beneficial. In this implementation of PPO, each update to the network parameters is calculated using gradients based upon a single trajectory. However, if multiple trajectories starting from different initial conditions but under the same policy and value networks were to be collected, the quality of each gradient update may be improved. Different network architectures, observation variables, and, if necessary, RL algorithms should also be investigated.

REFERENCES

[1] NASA, "Solar Electric Propulsion: Developing solar electric power and propulsion systems for efficient travel beyond low Earth orbit," Electronically, 2016.

[2] P. Enright and B. A. Conway, "Discrete approximations to optimal trajectories using direct transcription and Nonlinear Programming," *Journal of Guidance Control and Dynamics*, Vol. 15, 09 1992, 10.2514/3.20934.

[3] J. Sims and S. Flanagan, "Preliminary design of low-thrust interplanetary missions," 1999.

[4] T. J. Debban, T. T. McConaghy, and J. M. Longuski, "Design and Optimization of Low-Thrust Gravity-Assist Trajectories to Selected Planets," *American Institute of Aeronautics and Astronautics*, 2002.

[5] J. Sims, P. Finlayson, E. Rinderle, M. Vavrina, and T. Kowalkowski, "Implementation of a low-thrust trajectory optimization algorithm for preliminary design," *AIAA/AAS Astrodynamics specialist conference and exhibit*, 2006, p. 6746.

[6] J. Englander and B. Conway, "Automated Solution of the Low-Thrust Interplanetary Trajectory Problem," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 1, 2017, pp. 15–27, 10.2514/1.G002124.

[7] J. D. Aziz, D. Scheeres, and G. Lantoine, *Differential Dynamic Programming in the Three-Body Problem*, 10.2514/6.2018-2223.

[8] J. Aziz, D. Scheeres, J. Parker, and J. Englander, "A Smoothed Eclipse Model for Solar Electric Propulsion Trajectory Optimization," *TRANSACTIONS OF THE JAPAN SOCIETY FOR AERONAUTICAL AND SPACE SCIENCES, AEROSPACE TECHNOLOGY JAPAN*, 2019, pp. 17–181.

[9] J. D. Aziz, J. S. Parker, D. J. Scheeres, and J. A. Englander, "Low-Thrust Many-Revolution Trajectory Optimization via Differential Dynamic Programming and a Sundman Transformation," *The Journal of the Astronautical Sciences*, Vol. 65, Jun 2018, pp. 205–228, 10.1007/s40295-017-0122-8.

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

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, Vol. 521, No. 7553, 2015, pp. 436–444.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, Vol. 529, No. 7587, 2016, pp. 484–489.

[13] B. Dachwald, "Evolutionary neurocontrol: a smart method for global optimization of low-thrust trajectories," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 2004, p. 5405.

[14] P. A. Witsberger and J. M. Longuski, "Interplanetary Trajectory Design using a Recurrent Neural Network and Genetic Algorithm: Preliminary Results," *AAS/AIAA Astrodynamics Specialist Conference, Snowbird, UT*, American Astronautical Society, Univelt, Aug. 2018.

[15] S. De Smet and D. J. Scheeres, "Identifying heteroclinic connections using artificial neural networks," *Acta Astronautica*, Vol. 161, 2019, pp. 192–199.

[16] X. Chu, K. T. Alfriend, J. Zhang, and Y. Zhang, "Q-Learning Algorithm for Path-Planning to Maneuver through a Satellite Cluster," *AAS/AIAA Astrodynamics Specialist Conference, Snowbird, UT*, American Astronautical Society, Univelt, Aug. 2018.

[17] B. Gaudet, R. Linares, and R. Furfaro, "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing," *CoRR*, Vol. abs/1810.08719, 2018.

[18] J. Broida and R. Linares, "Spacecraft Rendezvous Guidance in Cluttered Environments via Reinforcement Learning," *29th AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI*, American Astronautical Society, Univelt, Jan. 2019. AAS 19-462.

[19] R. Furfaro, I. Boise, M. Orlandelli, P. Di Lizia, F. Tupputo, and R. Linares, "Deep Learning for Autonomous Lunar Landing," 2018.

[20] D. Miller and R. Linares, "Low-Thrust Optimal Control via Reinforcement Learning," *29th AAS/AIAA Space Flight Mechanics Meeting, Ka'anapali, HI*, American Astronautical Society, Univelt, Jan. 2019. AAS 19-560.

[21] D. H. Ellison, B. A. Conway, J. A. Englander, and M. T. Ozimek, "Analytic Gradient Computation for Bounded-Impulse Trajectory Models Using Two-Sided Shooting," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 7, 2018, pp. 1449–1462, 10.2514/1.G003077.

[22] D. H. Ellison, B. A. Conway, J. A. Englander, and M. T. Ozimek, "Application and Analysis of Bounded-Impulse Trajectory Models with Analytic Gradients," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 8, 2018, pp. 1700–1714, 10.2514/1.G003078.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, Vol. abs/1707.06347, 2017.

[24] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *CoRR*, Vol. abs/1502.05477, 2015.

[25] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, Vol. abs/1412.6980, 2014.