Reinforcement Learning for Low-Thrust Trajectory Design of Interplanetary Missions

Preprint · August 2020

CITATIONS
0

READS
117

2 authors:

Alessandro Zavoli
Sapienza University of Rome
60 PUBLICATIONS **202** CITATIONS
SEE PROFILE

Lorenzo Federici
Sapienza University of Rome
27 PUBLICATIONS **65** CITATIONS
SEE PROFILE

AAS 20-616

REINFORCEMENT LEARNING FOR LOW-THRUST TRAJECTORY DESIGN OF INTERPLANETARY MISSIONS

Alessandro Zavoli* **and Lorenzo Federici†**

This paper investigates the use of Reinforcement Learning for the robust design of low-thrust interplanetary trajectories in presence of severe disturbances, modeled alternatively as Gaussian additive process noise, observation noise, control actuation errors on thrust magnitude and direction, and possibly multiple missed thrust events. The optimal control problem is recast as a time-discrete Markov Decision Process to comply with the standard formulation of reinforcement learning. An open-source implementation of the state-of-the-art algorithm Proximal Policy Optimization is adopted to carry out the training process of a deep neural network, used to map the spacecraft (observed) states to the optimal control policy. The resulting Guidance and Control Network provides both a robust nominal trajectory and the associated closed-loop guidance law. Numerical results are presented for a typical Earth-Mars mission. First, in order to validate the proposed approach, the solution found in a (deterministic) unperturbed scenario is compared with the optimal one provided by an indirect technique. Then, the robustness and optimality of the obtained closed-loop guidance laws is assessed by means of Monte Carlo campaigns performed in the considered uncertain scenarios. These preliminary results open up new horizons for the use of reinforcement learning in the robust design of interplanetary missions.

INTRODUCTION

In recent years, the possibility of using small or micro-spacecraft in interplanetary missions is drawing the attention of scientists and engineers around the world interested in reducing both development time and cost of the mission, without affecting significantly its scientific return. The first deep-space micro-spacecraft, PROCYON,[1] was developed in little more than a year in 2014 by the University of Tokyo and JAXA, at a very low cost if compared to standard-size spacecraft. Despite the malfunctioning of the main thruster, the PROCYON mission has been ubiquitously called a success, paving the way for similar mission concepts by other space agencies. In 2018, NASA released the first two interplanetary CubeSats, part of the MarCO (Mars Cube One) mission,[2] which successfully accomplished their goal of providing a real-time communication link to Earth during the entry, descent, and landing phase of InSight lander. The same year, ESA's first stand-alone CubeSat mission for deep-space, M–Argo (Miniaturised – Asteroid Remote Geophysical Observer) has been announced,[3] and it is likely to be ready for launch in mid-2021 at the earliest.

Low-thrust electric propulsion is a key technology for enabling small/micro-satellite interplanetary missions, as it provides the spacecraft with significantly lower specific propellant consumption.

*Researcher, Department of Mechanical and Aerospace Engineering, Sapienza University of Rome, via Eudossiana 18, 00184, Rome, Italy

†PhD student, Department of Mechanical and Aerospace Engineering, Sapienza University of Rome, via Eudossiana 18, 00184, Rome, Italy

However, because of the limited budget, micro-spacecraft generally mount components with a low technological readiness level. This increases the risk of incurring unexpected control execution errors and/or missed thrust events (MTEs) during any of the long thrusting periods. In addition, small spacecraft have limited ground station access, and larger uncertainties in the state knowledge (i.e., in the observations for orbit determination) should be expected with respect to standard missions.

Typically, when designing the mission, the engineers take these uncertainties into account *a posteriori*,[4,5] by means of time-consuming iterative procedures which often bring to suboptimal solutions and over-conservative margins. This design methodology is particularly unsuitable for micro-spacecraft missions, where the possibility to have large propellant margins and system redundancy is almost completely excluded. In this respect, recent works attempted to address the robust design of interplanetary trajectories by using novel optimization techniques. As an example, the problem of designing optimal risk-aware trajectories, which guarantee the safety of the spacecraft when it operates in uncertain environments, was addressed by applying chance-constrained optimal control,[6] combined with a convex optimization approach, to deal with impulsive maneuvers,[7] or with a direct/indirect hybrid optimization method, to deal with continuous-thrust.[8] Stochastic Differential Dynamic Programming (SDDP) was applied to interplanetary trajectory design in presence of Gaussian-modeled state uncertainties.[9,10] Also, the robust design of a low-thrust interplanetary transfer to a near-Earth asteroid was performed by using evidence theory to model epistemic uncertainties in the performance of the main thruster and in the magnitude of the departure hyperbolic excess velocity.[11] Belief-based transcription procedures for the stochastic optimal control problem were proposed for the robust design of space trajectories under stochastic and epistemic uncertainties,[12,13] incorporating also navigation analysis in the formulation to update the knowledge of the spacecraft state in presence of observations.[14]

Deep Learning in Spaceflight Mechanics

The interest in the application of deep learning techniques to optimally and robustly solve control problems is rapidly increasing in recent years, especially for space applications. In this context, the term G&CNet (namely, Guidance and Control Network) was coined at the European Space Agency[15] to refer to an on-board system that provides real-time guidance and control functionalities to the spacecraft by means of a Deep Neural Network (DNN) that replaces traditional control and guidance architectures. DNNs are among the most versatile and powerful machine learning tools, thanks to their unique capability of accurately approximating complex, nonlinear input-output functions, provided that a sufficiently large amount of data (training set) consisting of sample input-output pairs is available.[16] Two alternative, and quite different, approaches can be used for training a G&CNet to solve an optimal control problem (OCP), depending on what training data are used and how they are collected.

In *Behavioral Cloning* (BC), given a set of trajectories from an expert (that is, labeled observations-controls pairs), the network is trained to reproduce (or clone) the expert behavior. Usually, these trajectories are obtained as the solution of a (deterministic) optimal control problem with randomized boundary conditions. Behavioral cloning has been successfully used to train a fast-execution G&CNet to control a spacecraft during a fuel-optimal low-thrust Earth-Venus transfer[17] as well as during a landing maneuver in a simplified dynamical model.[18] This approach proved to be computationally efficient, and it benefits from state-of-the-art implementations of supervised learning algorithms.[19] However, it shows a number of downsides that make it unsuitable for robust trajectory design. In fact, the BC effectiveness rapidly worsens when the G&CNet is asked to solve problems

that fall outside of the set of expert demonstrations it was trained in. As a consequence, when dealing with Stochastic Optimal Control Problems (SOCPs), a drop in performance (or even divergence) may occur when, because of uncertainty, the flight trajectory starts moving away from the training set domain, typically populated by solutions coming from deterministic OCPs. To recover a correct behavior, a DAGGER (Dataset Aggregation) algorithm can be used. In this case, the solution process features and additional loop where new training data are provided "on-line" by an expert (e.g., an OCP solver) as they are required to cover previously unknown situations. This approach has been effectively exploited to improve the network accuracy in controlling a lander during a powered descent on the Lunar surface.[20] However, the effectiveness of BC for robust trajectory design remains doubtful, especially when solutions from deterministic OCPs are used as expert demonstrations. Recently, an attempt has been performed to train a network by BC with a training set encompassing trajectories perturbed by random MTEs,[21] showing promising results. However, the possibility of having other types of state and control uncertainties has not been addressed yet.

A different approach is represented by *Reinforcement Learning* (RL), which involves learning from experience rather than from expert demonstrations. In RL, a software agent (e.g., the G&CNet) autonomously learns how to behave in a (possibly) unknown dynamical environment, modeled as a Markov Decision Process (MDP), so as to maximize some utility-based function that plays the role of the merit index in traditional optimal control problems. Differently from the BC approach, there is no pre-assigned data set of observations-controls pairs to learn from, so the agent is not told in advance what actions to take in a given set of states. Instead, the agent is left free to explore the environment, by repeatedly interacting with a sufficiently large number of realizations of it. The only feedback the agent receives back is a numerical reward collected at each time step, which helps the agent understanding how good or how bad its current performance is. In this framework, the final goal of the RL-agent is to learn the control policy that maximizes the expected cumulative sum of rewards over a trajectory. Because MDP allows only scalar reward functions, a careful choice, or shaping, of the reward is mandatory to efficiently guide the agent during training, while ensuring compliance with (any) problem constraints. Deep RL methods have obtained promising results in a number of spaceflight dynamics problems, such as low-thrust interplanetary trajectory design,[22–24] 3-DoF and 6-DoF landing guidance with application to a powered descent,[25] trajectory optimization in the cislunar environment,[26,27] and the design of guidance algorithms for rendezvous and docking maneuvers.[28,29]

This paper aims at investigating the use of Reinforcement Learning for the robust design of a low-thrust interplanetary trajectory in presence of uncertainty. Specifically, uncertainties on the spacecraft state, caused by unmodeled dynamical effects, on orbit determination, because of inaccurate knowledge, and on the applied control, due to execution errors and missed thrust events, will be considered in the present analysis. RL has been selected as optimization algorithm since it has the clear advantage of not requiring the *a priori* generation of any optimal trajectory to populate the training set, as data are gathered by running directly the current best found control policy on the stochastic environment. In this way, the agent is able to progressively improve, in an autonomous way, the performance and robustness of its control policy, in order to achieve the mission goals regardless of the uncertainties that may arise. This feature makes RL the ideal candidate to solve the problem at hand. At present, most of the research encompassing RL for spacecraft trajectory design deals exclusively with deterministic environments. Thus, one of the main contributions of this paper is the investigation of the possible extension of RL applicability also to stochastic scenarios.

The paper is organized as follows. First, the optimization problem is formulated as a Markov

Decision Process, and the mathematical models used to describe the state, observation, and control uncertainties acting on the system are defined. The expression of the reward function, which includes both the merit index and the problem constraints (e.g., fixed final spacecraft position and velocity), is given as well. Next, after a brief introduction of the basic concepts and notation of Reinforcement Learning, the RL algorithm used in this work, named Proximal Policy Optimization, is described in detail. Furthermore, the configuration selected for the DNN and the values used for the algorithm hyper-parameters are reported. Then, numerical results are presented for the case study of the paper, that is, a time-fixed low-thrust Earth-Mars rendezvous mission. Specifically, the effect of each source of uncertainties on the system dynamics is analysed independently and the obtained results are compared in terms of trajectory robustness and optimality. Eventually, the reliability of the obtained solutions is assessed by means of Monte Carlo simulations. A section of conclusions ends the paper.

PROBLEM STATEMENT

This paper investigates the use of RL algorithms for the design of robust low-thrust interplanetary trajectories. For the sake of comparison with other research papers,[9,10] a three-dimensional time-fixed minimum-fuel Earth-Mars rendezvous mission is considered as a test case. The spacecraft leaves the Earth with zero excess of hyperbolic velocity, and it is assumed to move in a Keplerian dynamical model under the sole influence of the Sun. The mission goal is to match Mars position and velocity at final time, with minimum propellant consumption. The values of the initial position $\boldsymbol{r}_{\oplus}$ and velocity $\boldsymbol{v}_{\oplus}$ of the Earth, the final position $\boldsymbol{r}_{\sigma}$ and velocity $\boldsymbol{v}_{\sigma}$ of Mars, the total transfer time t_f, the initial spacecraft mass m_0, and the spacecraft engine parameters (maximum thrust T_{max} and effective exhaust velocity u_{eq}) are the same as in the paper by Lantoine and Russell,[30] and are reported in Table 1. In all simulations, the physical quantities have been made non-dimensional by using as reference values the Earth-Sun mean distance $\bar{r} = 149.6 \times 10^6\,\mathrm{km}$, the corresponding circular velocity $\bar{v} = \sqrt{\mu_{\odot}/\bar{r}}$, and the initial spacecraft mass $\bar{m} = m_0$.

Table 1: Problem data.

Variable	Value
N	40
t_f, days	358.79
T_{max}, N	0.50
u_{eq}, km/s	19.6133
m_0, kg	1000
$\mu_{\odot}$, $\mathrm{km}^3/\mathrm{s}^2$	132712440018
$\boldsymbol{r}_{\oplus}$, km	$[-140699693,\ -51614428,\ 980]^T$
$\boldsymbol{v}_{\oplus}$, km/s	$[9.774596,\ -28.07828,\ 4.337725 \times 10^{-4}]^T$
$\boldsymbol{r}_{\sigma}$, km	$[-172682023,\ 176959469,\ 7948912]^T$
$\boldsymbol{v}_{\sigma}$, km/s	$[-16.427384,\ -14.860506,\ 9.21486 \times 10^{-2}]^T$

The stochastic effects here considered are *state uncertainties*, which refer to the presence of unmodeled dynamics, *observation uncertainties*, related to measurement noise and/or inaccuracies in orbital determination that lead to imperfect knowledge of the spacecraft state, and *control uncertainties*, which account for both random actuation errors (i.e., in the direction and magnitude of the thrust), and *single* or *multiple MTEs*, which correspond to null thrust occurrences.

Markov Decision Process

Let us briefly introduce the mathematical formulation of a generic Markov Decision Process (MDP), which is required to properly setup the mathematical framework of deep RL algorithms. Let $\boldsymbol{s}_k \in S \subset \mathbb{R}^n$ be a vector that completely identifies the *state* of the system (e.g., the spacecraft) at time t_k. In general, the complete system state at time t_k is not available to the controller, which instead relies on an *observation* vector $\boldsymbol{o}_k \in O \subset \mathbb{R}^m$. Observations might be affected by noise or uncertainty, and are thus written as a function of a random vector $\boldsymbol{\omega}_{o,k} \in \Omega_o \subset \mathbb{R}^{m_w}$. The commanded *action* $\boldsymbol{a}_k$ at time t_k is the output of a state-feedback control policy $\pi : O \rightarrow A$, that is: $\boldsymbol{a}_k = \pi(\boldsymbol{o}_k) \in A \subset \mathbb{R}^l$. The actual *control* $\boldsymbol{u}_k \in A$ differs from the commanded action due to possible execution errors, modeled as a function of a stochastic control disturbance vector $\boldsymbol{\omega}_{a,k} \in \Omega_a \subset \mathbb{R}^{l_w}$. A stochastic, time-discrete dynamical model f is considered for the system state. The uncertainty on the system dynamics at time t_k is modeled as a random vector $\boldsymbol{w}_{s,k} \in \Omega_s \subset \mathbb{R}^{n_w}$. As a result, the dynamical system evolution over time is described by the following equations:

$$\boldsymbol{s}_{k+1} = f(\boldsymbol{s}_k, \boldsymbol{u}_k, \boldsymbol{\omega}_{s,k}) \tag{1}$$

$$\boldsymbol{o}_k = h(\boldsymbol{s}_k, t_k, \boldsymbol{\omega}_{o,k}) \tag{2}$$

$$\boldsymbol{u}_k = g(\boldsymbol{a}_k, \boldsymbol{\omega}_{a,k}) \tag{3}$$

$$\boldsymbol{a}_k = \pi(\boldsymbol{o}_k) \tag{4}$$

The problem goal is to find the optimal control policy π^* that maximizes the expected value of the discounted sum of rewards, that, in an episodic form, is:

$$J = \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{k=0}^{N-1} \gamma^k R(\boldsymbol{s}_k, \boldsymbol{u}_k, \boldsymbol{s}_{k+1}) \right] \tag{5}$$

where $R(\boldsymbol{s}_k, \boldsymbol{u}_k, \boldsymbol{s}_{k+1})$ is the reward associated with transitioning from state $\boldsymbol{s}_k$ to state $\boldsymbol{s}_{k+1}$ due to control $\boldsymbol{u}_k$, $\gamma \in (0, 1]$ is a discount factor that is used to either encourage long term planning ($\gamma = 1$) or short term rewards ($\gamma \ll 1$), and N is the number of steps in one episode. Note that $\mathbb{E}_{\tau \sim \pi}$ here denotes the expectation taken over a trajectory τ, that is, a sequence of state-action pairs $\tau = \{(\boldsymbol{s}_0, \boldsymbol{a}_0), \ldots (\boldsymbol{s}_{N-1}, \boldsymbol{a}_{N-1})\}$ sampled according to the closed-loop dynamics in Eqs. (1)-(4).

Note that, in an episodic setting, $J = V^\pi(\boldsymbol{s}_0)$, being $V^\pi(\boldsymbol{s}_k)$ the value function, defined as the expected return obtained by starting from state $\boldsymbol{s}_k$ and acting according to policy π until the end of the episode:

$$V^\pi(\boldsymbol{s}_k) = \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{k'=k}^{N-1} \gamma^{k'} R(\boldsymbol{s}_{k'}, \boldsymbol{u}_{k'}, \boldsymbol{s}_{k'+1}) \right] \tag{6}$$

Formulating an Earth-Mars Mission as a Markov Decision Process

This general model is now specified for the Earth-Mars transfer problem at hand. During the mission, the spacecraft state $\boldsymbol{s}_k$ at any time step $t_k = k\, t_f/N$, $k \in [0, N]$, is identified by its inertial position $\boldsymbol{r}$ and velocity $\boldsymbol{v}$ with respect to Sun, and by its total mass m:

$$\boldsymbol{s}_k = \left[\boldsymbol{r}_k^T, \boldsymbol{v}_k^T, m_k\right]^T \in \mathbb{R}^7 \tag{7}$$

The low-thrust trajectory is approximated as a series of ballistic arcs connected by impulsive ΔVs, similarly to what done in the well-known Sims-Flanagan model.[31] The magnitude of the k-th impulse is limited by the amount of ΔV that could be accumulated over the corresponding trajectory

segment by operating the spacecraft engine at maximum thrust T_{max}:

$$\Delta V_{max,k} = \frac{T_{max}}{m_k}\frac{t_f}{N} \tag{8}$$

So, the commanded action at time t_k corresponds to an impulsive ΔV:

$$\boldsymbol{a}_k = \Delta \boldsymbol{V}_k \in [-\Delta V_{max,k}, \Delta V_{max,k}]^3 \subset \mathbb{R}^3. \tag{9}$$

Since the spacecraft moves under Keplerian dynamics between any two time steps, in a deterministic scenario the spacecraft state can be propagated analytically with a closed-form transition function:

$$\begin{bmatrix} \boldsymbol{r}_{k+1} \\ \boldsymbol{v}_{k+1} \\ m_{k+1} \end{bmatrix} = f(\boldsymbol{r}_k, \boldsymbol{v}_k, m_k, \Delta \boldsymbol{V}_k) = \begin{bmatrix} \hat{f}_k \boldsymbol{r}_k + \hat{g}_k(\boldsymbol{v}_k + \Delta \boldsymbol{V}_k) \\ \dot{\hat{f}}_k \boldsymbol{r}_k + \dot{\hat{g}}_k(\boldsymbol{v}_k + \Delta \boldsymbol{V}_k) \\ m_k \exp\left(-\frac{|\Delta \boldsymbol{V}_k|}{u_{eq}}\right) \end{bmatrix} \tag{10}$$

where $\hat{f}_k$ and $\hat{g}_k$ are the Lagrange coefficients at k-th step, defined as in Ref. 32, and the mass update is obtained through Tsiolkovsky equation.

At time t_f, the final ΔV is calculated so as to match Mars velocity, that is:

$$\Delta \boldsymbol{V}_N = \min\left(|\boldsymbol{v}_{\mathcal{O}} - \boldsymbol{v}_N|, \Delta V_{max,N}\right) \frac{\boldsymbol{v}_{\mathcal{O}} - \boldsymbol{v}_N}{|\boldsymbol{v}_{\mathcal{O}} - \boldsymbol{v}_N|} \tag{11}$$

and the final spacecraft state is evaluated as:

$$\boldsymbol{r}_f = \boldsymbol{r}_N \tag{12}$$

$$\boldsymbol{v}_f = \boldsymbol{v}_N + \Delta \boldsymbol{V}_N \tag{13}$$

$$m_f = m_N \exp\left(-|\Delta \boldsymbol{V}_N|/u_{eq}\right) \tag{14}$$

The (deterministic) observations collected at time t_k are:

$$\boldsymbol{o}_k = \left[\boldsymbol{r}_k^T, \boldsymbol{v}_k^T, m_k, t_k\right]^T \in \mathbb{R}^8 \tag{15}$$

The value selected for the total number of time steps N is reported in Table 1.

State Uncertainties. For the sake of simplicity, uncertainties on the spacecraft dynamics are modeled as additive Gaussian noise on position and velocity at time t_k, $k \in (0, N]$, that is:

$$\boldsymbol{w}_{s,k} = \begin{bmatrix} \delta \boldsymbol{r}_k \\ \delta \boldsymbol{v}_k \end{bmatrix} \sim \mathcal{N}(\boldsymbol{0}_6, \boldsymbol{R}_{s,k}) \in \mathbb{R}^6 \tag{16}$$

where $\boldsymbol{R}_{s,k} = \text{diag}\left(\sigma_r^2 \boldsymbol{I}_3, \sigma_v^2 \boldsymbol{I}_3\right)$ is the covariance matrix, $\boldsymbol{I}_n$ (respectively, $\boldsymbol{0}_n$) indicates an identity (respectively, null) matrix with dimension $n \times n$ (respectively, $n \times 1$), and σ_r, σ_v are the standard deviations on position and velocity. So, the stochastic dynamical model is written as:

$$\begin{bmatrix} \boldsymbol{r}_{k+1} \\ \boldsymbol{v}_{k+1} \\ m_{k+1} \end{bmatrix} = f(\boldsymbol{r}_k, \boldsymbol{v}_k, m_k, \boldsymbol{u}_k) + \begin{bmatrix} \delta \boldsymbol{r}_{k+1} \\ \delta \boldsymbol{v}_{k+1} \\ 0 \end{bmatrix} \tag{17}$$

Observation Uncertainties. The uncertainty in the knowledge of spacecraft position and velocity due to errors in the orbital determination is modeled as additive Gaussian noise on the deterministic observations at time t_k:

$$\boldsymbol{o}_k = \begin{bmatrix} \boldsymbol{r}_k \\ \boldsymbol{v}_k \\ m_k \\ t_k \end{bmatrix} + \begin{bmatrix} \delta\boldsymbol{r}_{o,k} \\ \delta\boldsymbol{v}_{o,k} \\ 0 \\ 0 \end{bmatrix} \tag{18}$$

being:

$$\boldsymbol{w}_{o,k} = \begin{bmatrix} \delta\boldsymbol{r}_{o,k} \\ \delta\boldsymbol{v}_{o,k} \end{bmatrix} \sim \mathcal{N}(\mathbf{0}_6, \boldsymbol{R}_{s,k}) \in \mathbb{R}^6 \tag{19}$$

Control Uncertainties. Control execution errors are modeled as a small three-dimensional rotation of the commanded ΔV vector, defined by Euler angles $(\delta\phi, \delta\vartheta, \delta\psi)$, and a slight variation δu of its modulus. Random variables $\delta\phi, \delta\vartheta, \delta\psi$ and δu are assumed to be Gaussian, with standard deviations $\sigma_\phi, \sigma_\vartheta, \sigma_\psi$ and σ_u. So, the control disturbance vector at time t_k is:

$$\boldsymbol{w}_{a,k} = \begin{bmatrix} \delta\phi_k \\ \delta\vartheta_k \\ \delta\psi_k \\ \delta u_k \end{bmatrix} \sim \mathcal{N}(\mathbf{0}_4, \boldsymbol{R}_{a,k}) \in \mathbb{R}^4 \tag{20}$$

where $\boldsymbol{R}_{a,k} = \text{diag}\left(\sigma_\phi^2, \sigma_\vartheta^2, \sigma_\psi^2, \sigma_u^2\right)$ is the covariance matrix.

The actual control $\boldsymbol{u}$ can be written as a function of the commanded action $\boldsymbol{a}$ at time t_k, $k \in [0, N)$, as:

$$\boldsymbol{u}_k = g(\boldsymbol{a}_k, \boldsymbol{w}_{a,k}) = (1 + \delta u_k)\boldsymbol{A}_k\boldsymbol{a}_k \tag{21}$$

where the rotation matrix $\boldsymbol{A}_k$ is evaluated, under the small-angle assumption, as:

$$\boldsymbol{A}_k = \begin{bmatrix} 1 & -\delta\psi_k & \delta\vartheta_k \\ \delta\psi_k & 1 & -\delta\phi_k \\ -\delta\vartheta_k & \delta\phi_k & 1 \end{bmatrix} \tag{22}$$

It is worth noting that, although the control disturbance vector is Gaussian, the effect obtained on the applied control is definitively non-Gaussian and, for this reason, the solution methods in Ref. 9 and 10 may not be applicable.

Missed Thrust Events. Besides small control execution errors, the effect of one or more consecutive MTEs over the course of the mission is also investigated. The MTE is modeled as a complete lack of thrust, even when commanded, that occurs at a randomly chosen time $t_{\hat{k}} \in [0, N)$, so that $\boldsymbol{u}_{\hat{k}} = \mathbf{0}_3$. With some probability $1 - p_{mte}$ the miss-thrust is recovered and for the remaining steps it never happens again. Otherwise, the MTE persists for an additional time-step. This procedure is repeated, but the MTE may last at most n_{mte} successive time steps, that is, from $t_{\hat{k}}$ to $t_{\hat{k}+n_{mte}-1}$.

The values used for the standard deviations and the other uncertainty model parameters introduced so far are reported in Table 2.

Table 2: Uncertainty model parameters.

σ_r, km	σ_v, km/s	σ_ϕ, deg	σ_ϑ, deg	σ_ψ, deg	σ_u	p_{mte}	n_{mte}
1.0	0.05	1.0	1.0	1.0	0.05	0.1	3

Reward Function. The objective of the optimization procedure is to maximize the (expected) final mass of the spacecraft, while ensuring the compliance with terminal rendezvous constraints on position and velocity. For this reason, the reward r_k collected by the agent at time t_k, for $k \in (0, N]$, is defined as:

$$r_k = -\mu_k - \lambda_1 \, e_{u,k-1} - \lambda_2 \, e_{s,k} \tag{23}$$

where:

$$\mu_k = \Delta m_k = \begin{cases} m_{k-1} - m_k & \text{if } k < N \\ m_{N-1} - m_f & \text{if } k = N \end{cases} \tag{24}$$

$$e_{u,k} = \max\left(0, |\boldsymbol{u}_k| - \Delta V_{max,k}\right) \tag{25}$$

$$e_{s,k} = \begin{cases} 0 & \text{if } k < N \\ \max\left(0, \max\left(\frac{|\boldsymbol{r}_f - \boldsymbol{r}_\mathcal{O}|}{|\boldsymbol{r}_\mathcal{O}|}, \frac{|\boldsymbol{v}_f - \boldsymbol{v}_\mathcal{O}|}{|\boldsymbol{v}_\mathcal{O}|}\right) - \varepsilon\right) & \text{if } k = N \end{cases} \tag{26}$$

Here μ_k is the cost function, that is, the consumed propellant mass, $e_{u,k}$ is the violation of the constraint relative to the maximum ΔV magnitude admissible for that segment (see Eq. 8), and $e_{s,k}$ is the violation of the constraint acting on the final state of the spacecraft, up to a given tolerance $\varepsilon = 10^{-3}$. The penalty factors $\lambda_1 = 100$ and $\lambda_2 = 50$ are used in the present work.

REINFORCEMENT LEARNING

This section briefly outlines the mathematical framework of the RL method selected to tackle the problem outlined in the previous section. The RL algorithm adopted in this work is *Proximal Policy Optimization*,[33] which is a model-free policy-gradient actor-critic method, widely recognized for the high performance demonstrated on a number of continuous and high-dimensional control problems.

A model-free approach is particularly suited to design a robust trajectory. In fact, model-free RL methods do not rely on an *explicit* knowledge of the MDP (Eqs. (1)-(4)) for returning the optimal control policy π^*. This fact leaves us the freedom to use complex, and possibly non-analytical, expressions for the transition function f, control model g, observation model h, and/or disturbance distributions, without requiring any change in the algorithm.

Policy-gradient algorithms are a common choice in RL. The underlying idea is to directly learn the policy $\pi(\boldsymbol{s})$ that maximizes the performance index J. A stochastic policy is generally considered, as more robust and oriented toward exploration than deterministic ones. In this sense, $\pi(\boldsymbol{s})$ should be intended as a shorthand for $\pi(\boldsymbol{a}|\boldsymbol{s})$, that is, the probability of choosing action $\boldsymbol{a}$, conditioned by being the system in the state $\boldsymbol{s}$.

In order to cope with complex environments and possibly large (and continuous) state and/or action spaces, the policy π is usually modeled by a DNN with parameters (i.e., weights and biases) θ, and referred to as $\pi_\theta(\boldsymbol{a}|\boldsymbol{s})$ to make the dependence on the network parameters explicit. A typical DNN is composed by a sequence of *layers*, each made up of a number of basic units, called *neurons*. A Multi-Layer Perceptron (MLP) is adopted in the present work: each neuron receives input signals

from the units in the previous layer, generates an output signal as a linear combination of the inputs, elaborated through a (typically nonlinear) *activation function*, and sends this signal to the units in the next layer. More complex architectures, such as Recurrent Neural Networks (RNNs), are also used in deep RL to cope with partially-observable or non-Markov environments, but they are not considered in this work. In the case of a deterministic policy, the network directly outputs the action, given the current system state as input; instead, when a stochastic policy is considered, the network returns some parameters of the underlying probability distribution, such as mean and variance for a Gaussian distribution, which is the most frequently adopted. The actual action is then sampled from this probability distribution.

Once the network architecture (i.e., number of layers, layer density, and activation functions) is assigned, the problem reduces to the search of the parameters θ^* that maximize the merit index J:

$$\theta^* = \arg\max_{\theta} J(\theta) = \arg\max_{\theta} \left(\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{k=0}^{N-1} r_k \right] \right) \tag{27}$$

where $r_k = R(\boldsymbol{s}_k, \boldsymbol{u}_k, \boldsymbol{s}_{k+1})$ is the immediate reward collected after taking action $\boldsymbol{a}_k \sim \pi_\theta(\cdot|\boldsymbol{s}_k)$.

In order to solve Eq. (27), policy-gradient algorithms perform a (stochastic) gradient ascent update on θ, that is: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$, where α is the *learning rate*, which is a constant, or slowly decreasing, user-defined hyper-parameter that controls the step-size of the gradient update.

The *Policy Gradient Theorem*[34] is used to rewrite the gradient $\nabla_\theta J(\theta)$ in a more suitable form:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{k=0}^{N-1} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_k|\boldsymbol{s}_k)\, Q^{\pi_\theta}(\boldsymbol{s}_k, \boldsymbol{a}_k) \right] \tag{28}$$

where $Q^{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{k'=k}^{N-1} r_{k'} \mid \boldsymbol{s}_k = \boldsymbol{s}, \boldsymbol{a}_k = \boldsymbol{a} \right]$ is the Action-Value function, or simply Q-function, that is, the expected return obtained by starting in state $\boldsymbol{s}$, taking action $\boldsymbol{a}$ and then acting according to π_θ.

The Q-function might be estimated by a Monte-Carlo approach, using the average return over episode samples. While unbiased, this estimate has a high variance, which makes this approach unsuitable for practical purposes. An improved solution relies on approximating the Q-value function by a second DNN, leading to the so called Actor-Critic method. The two neural networks run in parallel and are concurrently updated: the Actor, which returns the parametrized policy π_θ, is updated by gradient ascent on the policy-gradient; the Critic, which returns the parametrized Q-function Q_ϕ, is recursively updated using Temporal Difference.[34] Intuitively, the Actor controls the agent behavior while the Critic evaluates the agent performance and gives a feedback to the Actor in order to efficiently update the policy.

To further improve the stability of the learning process and reduce variance in the sample estimate for the policy gradient, it is possible to subtract a function of the system state only, called baseline $b(\boldsymbol{s})$, from the expression (28) of the policy gradient, without changing it in expectation. The most common choice of baseline is the value function $V^{\pi_\theta}(\boldsymbol{s})$, which, subtracted from the Action-Value function $Q^{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a})$, leads to the definition of the advantage function $A^{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a}) = Q^{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a}) - V^{\pi_\theta}(\boldsymbol{s})$, which expresses by how much the total reward improves by taking a specific action $\boldsymbol{a}$ in state $\boldsymbol{s}$, instead of randomly selecting the action according to $\pi_\theta(\cdot|\boldsymbol{s})$. From a computational point of view, the advantage function is usually computed by using the so-called generalized advantage

estimation:

$$\hat{A}_k = \sum_{k'=k}^{N-1} (\gamma\lambda)^{k'-k} \delta_{k'} \tag{29}$$

where:

$$\delta_k = r_k + \gamma V^{\pi_\theta}(\boldsymbol{s}_{k+1}) - V^{\pi_\theta}(\boldsymbol{s}_k) \tag{30}$$

is the Temporal-Difference error, that is an unbiased estimate of the advantage function, being $\mathbb{E}_{\tau\sim\pi_\theta}[\delta_k] = A^{\pi_\theta}(\boldsymbol{s}_k, \boldsymbol{a}_k)$.

So, the policy gradient can be evaluated as:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{k=0}^{N-1} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_k|\boldsymbol{s}_k)\, \hat{A}_k\right] \tag{31}$$

This leads to the Advantage Actor-Critic (A2C) method,[35] whose basic architecture is reported in Fig. 1. In this case, the Critic returns a parameterized value function V_ϕ, necessary for the advantage estimation in Eqs. (29)–(30).

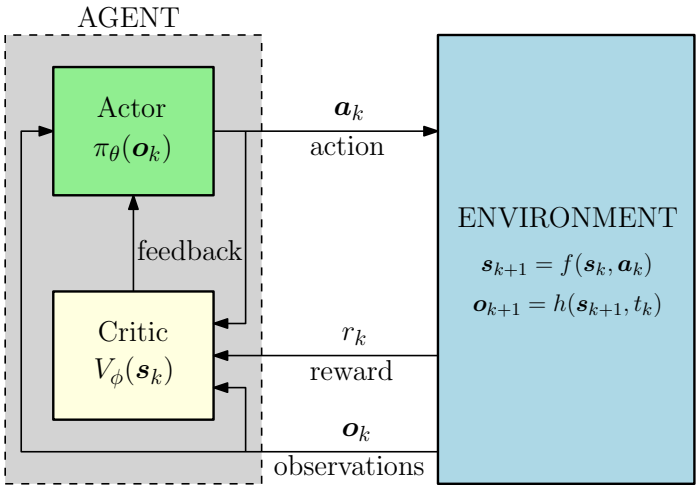


Figure 1: Schematic of the Advantage Actor-Critic RL process, for a deterministic MDP.

PPO can be seen as a further evolution of the (advantage) Actor-Critic method. In order to avoid too large policy updates when using Eq. (31) during gradient ascent, which usually cause performance collapse, PPO introduces a "clipped surrogate objective function" J^{clip} in place of J, to constrain the updated policy π_θ to stay in a small range ϵ, named clip range, around the previous value $\pi_{\theta_{old}}$, by clipping the probability that the new policy moves outside of the interval $[1-\epsilon, 1+\epsilon]$.

Let $\tilde{r}_k(\theta)$ be the probability ratio:

$$\tilde{r}_k(\theta) = \frac{\pi_\theta(\boldsymbol{a}_k|\boldsymbol{s}_k)}{\pi_{\theta_{old}}(\boldsymbol{a}_k|\boldsymbol{s}_k)} \tag{32}$$

The clipped surrogate objective function can be written as:

$$J^{clip}(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\frac{1}{N}\sum_{k=0}^{N-1} \min\left(\tilde{r}_k(\theta)\hat{A}_k, \text{clip}(\tilde{r}_k(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_k\right)\right] \tag{33}$$

Empirical evidences suggest that using a clip range ϵ that decreases linearly with the training steps improves the learning effectiveness. Typically, in PPO the policy and value function updates are carried out all at once by gathering in the set of parameters θ the weights and biases of both the Actor and Critic networks, and by including in the objective function a mean-squared value function error term H:

$$H(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\frac{1}{N}\sum_{k=0}^{N-1}\frac{1}{2}\left(V_\theta(\boldsymbol{s}_k) - \sum_{k'=k}^{N} r_{k'}\right)^2\right] \tag{34}$$

and an entropy term S:

$$S(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\frac{1}{N}\sum_{k=0}^{N-1}\mathbb{E}_{\boldsymbol{a}\sim\pi_\theta(\cdot|\boldsymbol{s}_k)}\left[-\log\pi_\theta(\boldsymbol{a}|\boldsymbol{s}_k)\right]\right] \tag{35}$$

which is usually added to ensure sufficient exploration, and prevent premature convergence to a suboptimal deterministic policy.[35] Eventually, the final surrogate objective function is:

$$J^{ppo}(\theta) = J^{clip}(\theta) - c_1 H(\theta) + c_2 S(\theta) \tag{36}$$

where c_1 and c_2 are two hyper-parameters, named value function coefficient and entropy coefficient, that control the relative importance of the various terms.

The overall learning process consists of two well-distinguished phases that are repeated iteratively: i) *policy rollout*, during which the current policy π_θ is run in n_{env} independent (e.g. parallel) realizations of the environment for n_b training episodes, collecting a set of $n_{env}n_b$ trajectories τ^i; ii) *policy update*, which is performed by running n_{opt} epochs of stochastic gradient ascent over n_b sampling mini-batches of on-policy data, that is, the data coming from the last rollout only. The algorithm terminates after a total number of training steps equal to T.

Remark. The above described framework has been derived in case of a (perfectly observable) Markov Decision Process. When a perfect knowledge of the state is not available or when the observations differ from the state, the same RL algorithm can be used, but in this case the Actor and Critic networks take as an input directly the observation $\boldsymbol{o}_k$.

Implementation Details

The results presented in this work have been obtained by using the PPO algorithm implementation by Stable Baselines,[36] an open-source library containing a set of improved implementations of RL algorithms based on OpenAI Baselines. The scientific library *pykep*,[37] developed at the European Space Agency, was instead used for the astrodynamics routines.

The selected G&CNet consists of two separate head networks, one for the control policy and the other for the value function, each one composed of two hidden layers. The G&CNet architecture is summarized in Table 3, which reports the number of neurons in each layer and the activation function used in each neuron. The tuning of PPO hyper-parameters was performed by using Optuna,[38] an open source hyper-parameter optimization framework for automated hyper-parameter search. The hyper-parameter optimization was realized on a deterministic version of the RL environment, with a budget of 500 trials, each with a maximum of 3×10^5 training steps. The optimal value of the hyper-parameters used in all simulations is reported in Table 4.

When dealing with constrained optimization problems, constraint violations are typically included as penalty terms inside the reward function (see Eq. (23)). In these cases, a penalty-based

Table 3: Network Configuration.

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

Table 4: PPO hyperparameters.

Hyper-parameter	Value
γ	0.9999
λ	0.99
α	$2.5 \times 10^{-4}\,(1 - t/T)^{\star}$
ϵ	$0.3\,(1 - t/T)^{\star}$
c_1	0.5
c_2	4.75×10^{-8}
n_{opt}	30

* t indicates the training step number

ε-constraint method, similar to those sometimes used in stochastic global optimization,[39] proved to be helpful to enforce constraints more gradually during optimization, allowing the agent to explore to a greater extent the solution space at the beginning of the training process. For this reason, as a modest original contribution of this paper, the constraint satisfaction tolerance ε also varies during the training, according to a (piecewise constant) decreasing trend:

$$\varepsilon = \begin{cases} 0.01 & \text{for the first } T/2 \text{ training steps} \\ 0.001 & \text{for the second } T/2 \text{ training steps} \end{cases} \tag{37}$$

NUMERICAL RESULTS

This section presents some preliminary results obtained by training the G&CNet on different environments (that is, mission scenarios), generated by considering separately each uncertainty source defined in the problem statement. In order to validate the proposed approach, the solution found in a (deterministic) unperturbed scenario is compared with the optimal one provided by an indirect technique. Then, the robustness and optimality of the obtained control policies is assessed by means of Monte Carlo campaigns performed in the considered uncertain scenarios.

Deterministic Optimal Trajectory

The ability of the presented methodology to deal with traditional, deterministic, optimal control problems is investigated first, by comparing the solution provided by the control policy π^{unp} trained in the deterministic (unperturbed) environment (see Eq. (10)), with the solution of the original Earth-Mars low-thrust transfer problem, found by using a well-established indirect optimization method used by the authors in other interplanetary transfers.[40] The two solutions are very close to each other in terms of trajectory and control direction. Also, the final mass of the RL solution ($600.23\,\mathrm{kg}$) is in good agreement with the (true) optimal mass obtained by the indirect method ($599.98\,\mathrm{kg}$). This slight difference is partly due to the fact that RL satisfies the terminal constraints with a lower accuracy (10^{-3} in RL vs 10^{-9} in the indirect method), and partly due to the (approximated) time-discrete, impulsive dynamical model adopted in the MDP transcription.

However, when applying RL to the solution of deterministic optimal control problems, two major downsides arise. First, terminal constraints cannot be explicitly accounted for, and constraint violations must be introduced in the reward function as (weighted) penalty terms. As a result, the accuracy on constraint satisfaction is generally looser than in traditional methods for solving optimal control problems. Second, RL is quite computationally intensive, even when applied to problems

as simple as the deterministic rendezvous mission here considered. This is mainly motivated by the fact that PPO is a model-free algorithm, hence, the knowledge of the underlying (analytical) dynamical model is not exploited at all. The only way the agent can obtain satisfactory results is to acquire as much experience (i.e., samples) as possible about the environment. In this respect, the solution of the deterministic problem here reported took about $2 \div 3$ hours (depending on the desired accuracy on the constraints) on a computer equipped with Intel Core i7-9700K CPU @$3.60\,\mathrm{GHz}$, while the indirect method just a few seconds.

Robust Trajectory Design

Besides the unperturbed, deterministic mission scenario (labeled unp), the following stochastic case-studies are considered: i) state uncertainties (st), ii) observation uncertainties (obs), iii) control uncertainties (ctr), iv) single missed thrust event ($mte, 1$), and v) multiple missed thrust events ($mte, 2$). Training the G&CNet in one of these environments leads to the definition of a corresponding policy, named π^{unp}, π^{st}, π^{obs}, π^{ctr}, $\pi^{mte,1}$, and $\pi^{mte,2}$, respectively. For each policy, the reference trajectory, which should be intended as robust to that source of uncertainty, is obtained by applying in the unperturbed environment a (deterministic) version of the policy (i.e., that always takes the action corresponding to the largest probability, instead of sampling from the probability distribution $\boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s})$), and recording the commands and spacecraft states along the flight.

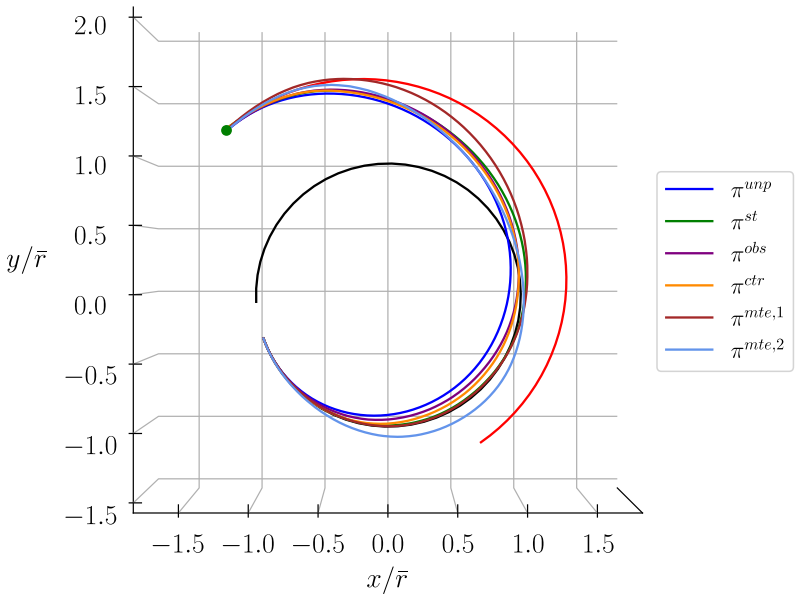


Figure 2: Earth-Mars trajectories corresponding to different robust policies. Differences with respect to the unperturbed policy trajectory are up-scaled by 5 for illustration purposes.

Figure 2 shows the robust trajectories obtained after a training that lasts up to $200\,\mathrm{Msteps}$, that roughly corresponds to $10 \div 12$ computing hours. For each case, only the best found solution during training (also accounting for the closed-loop behavior described in the next section) is reported. To add robustness, these trajectories tend to approach Mars orbit in advance with respect to the optimal solution, so as to maximize the probability of meeting the terminal constraints even in presence of late perturbations and/or control errors.

Table 5: Robust trajectory overview.

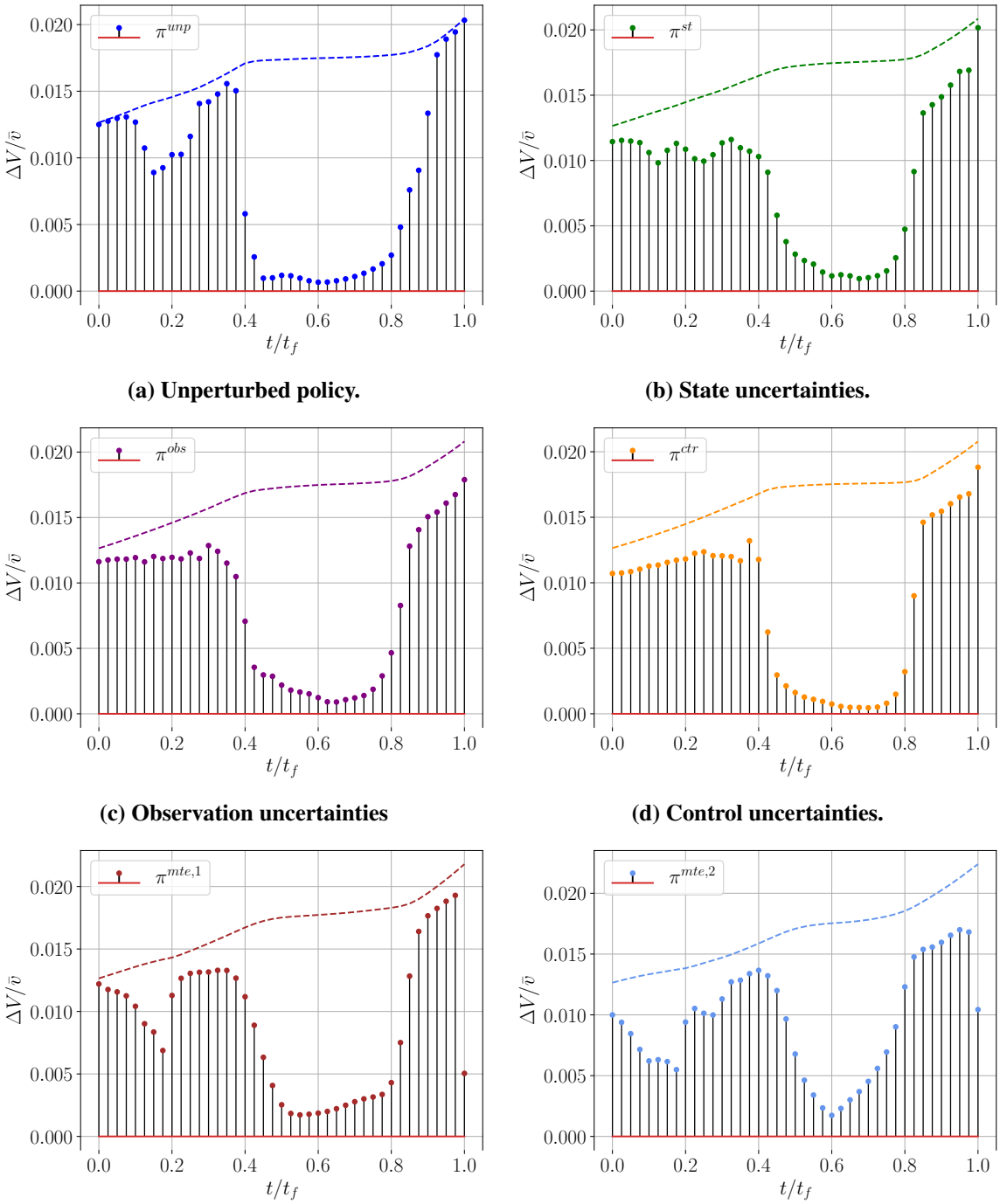
Policy	Settings			Results			
	T	n_{env}	n_b	m_f, kg	$\Delta r_f/r_{\mathcal{O}}$, ‰	$\Delta v_f/v_{\mathcal{O}}$, ‰	J
π^{unp}	$48M$	8	4	600.23	0.999	0	-0.3998
π^{st}	$112M$	8	4	588.41	0.767	0	-0.4116
π^{obs}	$200M$	8	4	591.48	0.782	0	-0.4085
π^{ctr}	$128M$	16	4	591.40	0.814	0	-0.4086
$\pi^{mte,1}$	$80M$	16	4	575.98	3.689	0	-0.5585
$\pi^{mte,2}$	$128M$	16	8	555.87	2.273	0	-0.5078

Table 5 summarizes the main features of these trajectories, that are, the final spacecraft mass m_f, constraint violations Δr_f and Δv_f, and cumulative reward J, as well as some environment-specific training settings. The solutions corresponding to a policy trained in a stochastic environment with perturbations on either state (π^{st}), observations (π^{obs}), or control direction and magnitude (π^{ctr}) satisfy the terminal constraints within the given tolerance (10^{-3}). In those cases, robustness is obtained by sacrificing less than $1 \div 2\%$ of the payload mass. Instead, the solutions $\pi^{mte,1}$ and $\pi^{mte,2}$, trained in the MTE environments, tend to slightly overcome Mars orbit, since they account, even in the unperturbed scenario, for the possible presence of MTEs, whose probability of occurrence during training has been exaggerated in this work for research purposes (at least one MTE must occur in any environment realization). Also, the final spacecraft mass obtained in these two cases is considerably worse than in the previous ones. In all presented cases, the error on the final velocity is zero. This result should not surprise the reader. In fact, the last ΔV is computed algebraically as a difference between the final spacecraft velocity and Mars velocity. Thus, the velocity constraint is automatically satisfied whenever this (computed) ΔV has a magnitude lower than the maximum admissible by the Sims-Flanagan model (see Eq. 11).

Figure 3 shows the distribution of the magnitude of the ΔVs over the flight time for the different robust trajectories. Dashed lines indicate the maximum allowed ΔV at each step, according to the Sims-Flanagan model (see Eq. (8)). As a general comment, the robust trajectories trained in the stochastic environments show a lower ΔV magnitude at the beginning and at the end of the transfer, with respect to the optimal deterministic solution π^{unp}, and, correspondingly, an higher magnitude in the central portion of the transfer. This sub-optimal distribution of the thrust is responsible for the additional propellant consumption of the robust solutions. Also, the applied ΔV is consistently lower than the maximum available in all cases except the unperturbed one, where an almost bang-off-bang pattern, which is the expected solution of this kind of optimal control problems, may be recognized. This is a distinctive feature of robust trajectories, that must satisfy the constraint on the maximum admissible value of ΔV, while leaving room for efficient correction maneuvers. As a final remark, in the two solutions obtained with policies $\pi^{mte,1}$ and $\pi^{mte,2}$ the last ΔV is considerably smaller than in the other solutions, probably because the two policies try to ensure the compliance with the final velocity constraint regardless of the possible presence of a MTE near the final time.

Closed-Loop Mission Analysis

The closed-loop performance of the G&CNets in their respective (stochastic) training environment are measured by using a Monte Carlo approach that consists in running each state-feedback policy in 500 randomly-generated environment realizations, or episodes. Figure 4 shows the re-

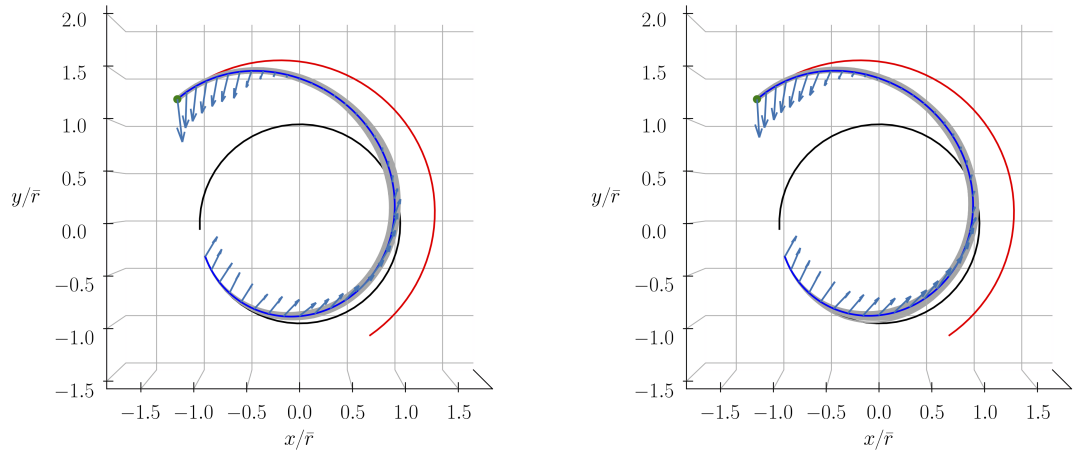


(a) Unperturbed policy. **(b) State uncertainties.**

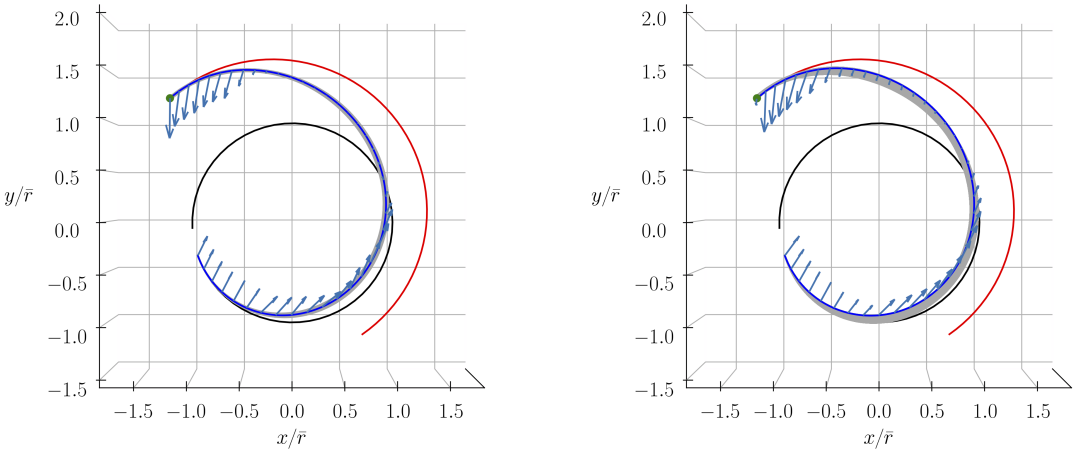
(c) Observation uncertainties **(d) Control uncertainties.**

(e) Single MTE. **(f) Multiple MTEs.**

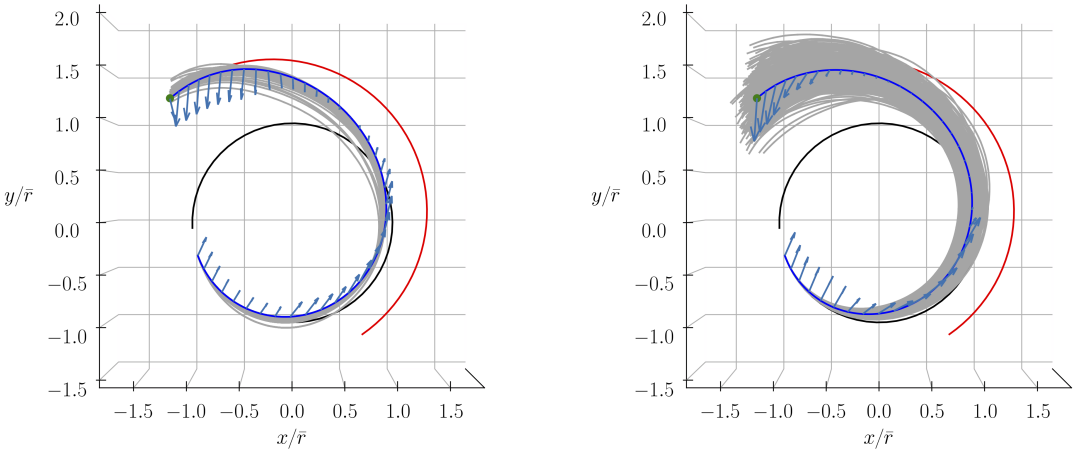
Figure 3: Magnitude of the ΔVs along the robust trajectories. Dashed lines indicate the maximum ΔV admitted at each time step by Sims-Flanagan model.



(a) π^{st} **(b)** π^{obs}



(c) π^{ctr} **(d)** $\pi^{mte,1}$



(e) $\pi^{mte,2}$ **(f)** π^{unp}

Figure 4: Monte Carlo simulations realized by running each policy in the correspondingly-perturbed environment, except for the unperturbed policy, which is run in the state-perturbed one. Differences are exaggerated by a factor 5 for illustration purposes.

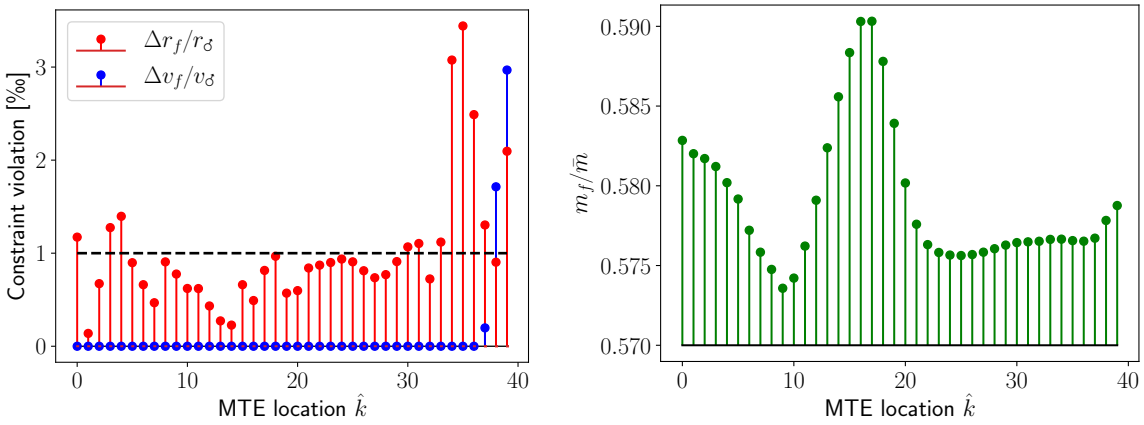
sults of the Monte Carlo campaigns in terms of spacecraft trajectory in each randomly-generated episode. Specifically, in each figure, the dark-blue line represents the robust trajectory, with the light-blue arrows indicating the ΔVs, while each gray line represents the trajectory obtained in one of the randomly-generated episodes. One may notice that, for policies π^{st}, π^{obs}, π^{ctr} and $\pi^{mte,1}$, the Monte Carlo generated trajectories have a greater dispersion in the central part of the mission, which tends to reduce, and disappear almost entirely, while approaching the final time, because of the imposed terminal constraints. This is not completely true for the case of multiple MTEs, where a small number of trajectories (15 out of 500) clearly miss the target by far.

Table 6: Results of the Monte Carlo simulations in terms of mean value (mean), standard deviation (std), and success rate (SR).

Policy	m_f, kg		$\Delta r_f/r_\mathcal{O}$, ‰		$\Delta v_f/v_\mathcal{O}$, ‰		SR, %
	mean	std	mean	std	mean	std	
π^{st}	578.04	9.48	0.742	0.322	0.0403	0.211	80.0
π^{obs}	580.94	9.53	0.807	0.343	0.0502	0.299	70.6
π^{ctr}	591.13	0.685	0.846	0.426	0.0620	0.270	69.0
$\pi^{mte,1}$	579.02	4.44	0.980	0.667	0.0987	0.438	69.8
$\pi^{mte,2}$	557.32	4.77	1.266	2.048	0.645	3.451	70.4

Table 6 shows that RL is able to cope with all these different stochastic scenarios effectively. Indeed, despite the severity of the considered perturbations/uncertainties, the success rate (that is, the percentage of solutions that meet final constraints within the prescribed accuracy) is rather high, over 70% in most cases and up to 80% when only additive Gaussian state perturbations are considered. For the sake of comparison, we also reported the results obtained by running policy π^{unp} in the state-perturbed stochastic environment (Fig. 4f). While the differences between the robust trajectories corresponding to policy π^{unp} and π^{st} seem minimal, the effects in the closed-loop simulations are apparent. Indeed, in none of the episodes the policy π^{unp} was able to reach the imposed accuracy on terminal state, while policy π^{st} succeeds in the 80% of the cases. Similar results are obtained when running the policy π^{unp} in any of the proposed perturbed (stochastic) environments. More precisely, the success rate is zero for both the state- and observation-uncertainty environments, while in case of control uncertainties on thrust magnitude/direction is 8%, and almost double in case of single (18.8%) or multiple (16.8%) MTEs.

The preliminary results found with policies $\pi^{mte,1}$ and $\pi^{mte,2}$ stressed that RL performance deteriorates substantially in the presence of MTEs. By looking at Figure 5a, it is clear that, in most cases (69.8% with a single MTE, 70.4% with multiple MTEs) the policy manages to recover from the complete absence of thrust, and meets the final constraints within the imposed tolerance (10^{-3}). However, in a few unfortunate scenarios, the MTE occurs in "crucial points" of the trajectory, that is, near final time, and the policy is not able to compensate for the missing ΔVs in any way. As a result, the terminal constraints cannot be met. This fact is confirmed by the high variance obtained on the constraint violation in both mission scenarios (single and multiple MTEs). Analogously, the drop in payload mass (Fig. 5b) highlights what are the most critical points in terms of thrust efficiency. The looser satisfaction of the terminal constraints is deemed responsible for the final rise in the achieved payload mass and might be misleading.



(a) Terminal constraint violation. **(b) Final spacecraft mass.**

Figure 5: Terminal constraint violation (a) and final spacecraft mass (b) obtained with policy $\pi^{mte,1}$ by varying the MTE location $\hat{k}$.

CONCLUSION

This paper presented a deep Reinforcement Learning (RL) framework to deal with the robust design of low-thrust interplanetary trajectories in presence of different sources of uncertainty. The stochastic optimal control problem must first be reformulated as a Markov Decision Process. Then, a state-of-the-art RL algorithm, named Proximal Policy Optimization (PPO), is adopted for the problem solution, and its prominent features over similar policy-gradient methods are outlined. Preliminary numerical results were reported for a three-dimensional Earth-Mars mission, by considering separately the effect of different types of uncertainties, namely, uncertainties on the dynamical model, on the observations, on the applied control, as well as the presence of a single or multiple, consecutive, missed thrust events.

The obtained results show the capability of PPO of solving simple interplanetary transfer problems, as the Earth-Mars mission here considered, in both deterministic and stochastic scenarios. The solution found in the deterministic case is in good agreement with the optimal solution provided by an indirect method. However, the high computational cost necessary to train the neural network discourages the use of a model-free RL algorithm in that circumstance. The power of RL becomes apparent when dealing with stochastic optimal control problems, where traditional methods are either cumbersome, impractical, or simply impossible to apply. Despite the reported results are only preliminary, the presented solutions seem very promising, in terms of both payload mass and constraint enforcement. The methodology here proposed is quite general and can be implemented, with the appropriate changes, to cope with a variety of spacecraft missions and uncertainty models. Also, extensions to arbitrary stochastic dynamical models (e.g., with possibly complex non-Gaussian perturbations) are straightforward. This is a major advantage with respect to other techniques presented in the literature based on ad-hoc extensions of traditional optimal control methods.

The preliminary results here proposed pave the way for reinforcement learning approaches in robust design of interplanetary trajectories. Additional work is obviously necessary in order to increase both the efficiency of the learning process and the reliability of the solutions. The high computational cost calls for the use of asynchronous algorithm, where the two processes of policy-

rollout (for collecting experience) and policy-update (for learning) run in parallel, so as to exploit at best the massive parallelization allowed by high performance computing clusters. Also, the use of recurrent neural networks should be investigated when dealing with non-Markov dynamical processes, as in the case of partial observability and multiple, correlated, missed thrust events. However, the most crucial point seems to be enhancing the constraint-handling capability of RL algorithms. The adoption of the ε-constraint relaxation is a modest contribution that goes in that direction. More advanced formulations of the problem, such as Constrained Markov Decision Process (CMDP), should be investigated in the future for this purpose.

REFERENCES

[1] S. Campagnola, N. Ozaki, Y. Sugimoto, C. H. Yam, H. Chen, Y. Kawabata, S. Ogura, B. Sarli, Y. Kawakatsu, R. Funase, *et al.*, "Low-thrust trajectory design and operations of PROCYON, the first deep-space micro-spacecraft," *25th International Symposium on Space Flight Dynamics, Munich, and 66th IAC, Jerusalem*, 2015.

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

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

[4] M. D. Rayman, T. C. Fraschetti, C. A. Raymond, and C. T. Russell, "Coupling of system resource margins through the use of electric propulsion: Implications in preparing for the Dawn mission to Ceres and Vesta," *Acta Astronautica*, Vol. 60, No. 10-11, 2007, pp. 930–938, https://doi.org/10.1016/j.actaastro.2006.11.012.

[5] F. E. Laipert and J. M. Longuski, "Automated missed-thrust propellant margin analysis for low-thrust trajectories," *Journal of Spacecraft and Rockets*, Vol. 52, No. 4, 2015, pp. 1135–1143, https://doi.org/10.2514/1.A33264.

[6] M. Ono, B. C. Williams, and L. Blackmore, "Probabilistic planning for continuous dynamic systems under bounded risk," *Journal of Artificial Intelligence Research*, Vol. 46, 2013, pp. 511–577, https://doi.org/10.1613/jair.3893.

[7] K. Oguri and J. W. McMahon, "Risk-aware Trajectory Design with Impulsive Maneuvers: Convex Optimization Approach," *AAS/AIAA Astrodynamics Specialist Conference*, Portland, Maine, 2019.

[8] K. Oguri and J. W. McMahon, "Risk-aware Trajectory Design with Continuous Thrust: Primer Vector Theory Approach," *AAS/AIAA Astrodynamics Specialist Conference*, Portland, Maine, 2019.

[9] N. Ozaki, S. Campagnola, R. Funase, and C. H. Yam, "Stochastic differential dynamic programming with unscented transform for low-thrust trajectory design," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 2, 2018, pp. 377–387, https://www.doi.org/10.2514/1.G002367.

[10] N. Ozaki, S. Campagnola, and R. Funase, "Tube Stochastic Optimal Control for Nonlinear Constrained Trajectory Optimization Problems," *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 4, 2020, pp. 645–655, https://www.doi.org/10.2514/1.G004363.

[11] M. Di Carlo, M. Vasile, C. Greco, and R. Epenoy, "Robust optimisation of low-thrust interplanetary transfers using evidence theory," *Advances in the Astronautical Sciences*, Vol. 168, 2019, pp. 339–358.

[12] C. Greco, M. Di Carlo, M. Vasile, and R. Epenoy, "An intrusive polynomial algebra multiple shooting approach to the solution of optimal control problems," *69th International Astronautical Congress*, 2018.

[13] C. Greco, M. Di Carlo, M. Vasile, and R. Epenoy, "Direct multiple shooting transcription with polynomial algebra for optimal control problems under uncertainty," *Acta Astronautica*, Vol. 170, 2020, pp. 224–234, https://doi.org/10.1016/j.actaastro.2019.12.010.

[14] C. Greco, S. Campagnola, and M. L. Vasile, "Robust Space Trajectory Design using Belief Stochastic Optimal Control," *AIAA Scitech 2020 Forum*, 2020, https://www.doi.org/10.2514/6.2020-1471.

[15] D. Izzo, M. Märtens, and B. Pan, "A survey on artificial intelligence trends in spacecraft guidance dynamics and control," *Astrodynamics*, Vol. 3, 2019, pp. 287–299, https://www.doi.org/10.1007/s42064-018-0053-6.

[16] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, Vol. 3, jan 1990, pp. 551–560, https://www.doi.org/10.1016/0893-6080(90)90005-6.

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

[18] C. Sanchez-Sanchez, D. Izzo, and D. Hennes, "Learning the optimal state-feedback using deep networks," *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2016, https://www.doi.org/10.1109/ssci.2016.7850105.

[19] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. Software available from tensorflow.org.

[20] R. Furfaro, I. Bloise, M. Orlandelli, P. Di Lizia, F. Topputo, R. Linares, *et al.*, "Deep learning for autonomous lunar landing," *AAS/AIAA Astrodynamics Specialist Conference*, Snowbird, UT, 2018.

[21] A. Rubinsztejn, R. Sood, and F. E. Laipert, "Neural Network Based Optimal Control: Resilience to Missed Thrust Events for Long Duration Transfers," *AAS/AIAA Astrodynamics Specialist Conference*, Portland, Maine, 2019.

[22] D. Miller and R. Linares, "Low-thrust optimal control via reinforcement learning," *Advances in the Astronautical Sciences*, Vol. 168, 2019, pp. 1817–1834.

[23] D. Miller, J. A. Englander, and R. Linares, "Interplanetary Low-Thrust Design Using Proximal Policy Optimization," *AAS/AIAA Astrodynamics Specialist Conference*, Portland, Maine, 2019.

[24] C. J. Sullivan and N. Bosanac, "Using Reinforcement Learning to Design a Low-Thrust Approach into a Periodic Orbit in a Multi-Body System," *AIAA Scitech 2020 Forum*, 2020, https://www.doi.org/10.2514/6.2020-1914.

[25] B. Gaudet, R. Linares, and R. Furfaro, "Deep reinforcement learning for six degree-of-freedom planetary landing," *Advances in Space Research*, Vol. 65, No. 7, 2020, pp. 1723–1741, https://www.doi.org/10.1016/j.asr.2019.12.030.

[26] A. Scorsoglio, R. Furfaro, R. Linares, and M. Massari, "Actor-critic reinforcement learning approach to relative motion guidance in near-rectilinear orbit," *Advances in the Astronautical Sciences*, Vol. 168, 2019, pp. 1737–1756.

[27] N. B. LaFarge, D. Miller, K. C. Howell, and R. Linares, "Guidance for Closed-Loop Transfers using Reinforcement Learning with Application to Libration Point Orbits," *AIAA Scitech 2020 Forum*, 2020, https://www.doi.org/10.2514/6.2020-0458.

[28] J. Broida and R. Linares, "Spacecraft rendezvous guidance in cluttered environments via reinforcement learning," *Advances in the Astronautical Sciences*, Vol. 168, 2019, pp. 1777–1788.

[29] K. Hovell and S. Ulrich, "On Deep Reinforcement Learning for Spacecraft Guidance," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, 2020, https://www.doi.org/10.2514/6.2020-1600.

[30] G. Lantoine and R. P. Russell, "A hybrid differential dynamic programming algorithm for constrained optimal control problems. part 2: Application," *Journal of Optimization Theory and Applications*, Vol. 154, No. 2, 2012, pp. 418–442, https://doi.org/10.1007/s10957-012-0038-1.

[31] J. Sims and S. Flanagan, "Preliminary design of low-thrust interplanetary missions," *AAS/AIAA Astrodynamics Specialist Conference*, 1999.

[32] R. R. Bate, D. D. Mueller, and J. E. White, *Fundamentals of Astrodynamics*. Dover, NY, 1971.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[34] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[35] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *International conference on machine learning*, 2016, pp. 1928–1937.

[36] A. Hill *et al.*, "Stable Baselines," `https://github.com/hill-a/stable-baselines`, 2018.

[37] D. Izzo, "PyKEP: open source tools for massively parallel optimization in astrodynamics (the case of interplanetary trajectory optimization)," *Proceedings of the 5th International Conference on Astrodynamics Tools and Techniques, ICATT*, 2012.

[38] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[39] L. Federici, B. Benedikter, and A. Zavoli, "EOS: a parallel, self-adaptive, multi-population evolutionary algorithm for constrained global optimization," *2020 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, IEEE, 2020.

[40] G. Colasurdo, A. Zavoli, A. Longo, L. Casalino, and F. Simeoni, "Tour of Jupiter Galilean moons: Winning solution of GTOC6," *Acta Astronautica*, Vol. 102, 2014, pp. 190–199, https://doi.org/10.1016/j.actaastro.2014.06.003.