Transformer-based Robust Feedback Guidance for Atmospheric Powered Landing

Jacopo Carradori*
Delft University of Technology, Delft, the Netherlands

Marco Sagliano †
German Aerospace Center, Bremen, Germany, 28359

Erwin Mooij‡
Delft University of Technology, Delft, the Netherlands

Rocket reusability is a key factor in enabling quicker and more cost-effective access to space. However, landing on Earth poses significant challenges due to the dynamic and highly uncertain environment. A robust Guidance, Navigation, and Control system is essential to guide the vehicle to the landing site while meeting terminal constraints and minimizing fuel consumption. This research integrates Meta-Reinforcement Learning with Gated Transformer XL Neural Networks to enhance the robustness of the powered guidance with respect to atmospheric and aerodynamic uncertainties, navigation and control errors, and dispersed initial conditions. By employing a 6-Degrees-of-Freedom dynamics model and accurate vehicle and environmental simulations, the agent learns a higher fidelity guidance policy compared to existing literature, demonstrating successful and robust performance in Monte Carlo simulations. In this complex scenario, the innovative attention-based neural networks also outperform recurrent neural networks, widely used for Reinforcement Learning-based space guidance applications.

I. Introduction

The evolving interest for space exploration and the increasing demands for cost-effective satellite launches brought a shift of focus toward the reusability of space vehicles, with particular attention to precise powered vertical landings.

A key element to achieve a successful landing is the Guidance, Navigation, and Control (GNC) system, which has the task of correctly driving the vehicle toward the landing site. It is fundamental to have a reliable, accurate, and robust guidance and control (G&C) policy that is also capable of running in real-time, within the computational constraints coming from the onboard computer. Solutions to this class of problems have been found using a problem formulation in vacuum [1, 2], with simplified aerodynamic models [3, 4], or using aerodynamic forces, but only for translational motion modeling [5, 6]. However, the solution is deeply modified by the presence of aerodynamic forces and torques, and a corresponding coupling between translational and rotational motion. Furthermore, using simplified models, such as an exponential atmospheric model, a constant drag, and a fixed center of mass, produces a solution, which considerably differs from the one actually flown. Finally, the wind can play a relevant role as well. Therefore, a full 6-Degrees-of-Freedom (DOF) problem formulation, together with accurate aerodynamic, atmospheric, and vehicle modeling, can be highly beneficial for the accuracy of the results obtained in the conceptual design phase.

The recent success of Machine Learning in natural language processing, image recognition, and robotics has raised a strong interest in investigating potential applications in the space sector. In this context, Reinforcement Learning (RL) is a methodology able to train agents to learn guidance and control policies. It consists of simulated interactions between an agent and an environment over a large number of episodes, mapping from observations to control actions using Neural Networks (NNs). It has been successfully employed in several spacecraft guidance problems, such as 6-DOF planetary landing on Mars [7], low-thrust transfer in cislunar environment [8], and proximity operations [9].

The use of Recurrent and Transformer NNs leverages their ability to capture long-term relationships and sequence dependencies, retaining information about temporal variations, to adapt to similar, slightly different, situations. This

*MSc Graduate, Section Astrodynamics and Space Mission. Currently: Young Graduate Trainee in Systems Engineering for a Mars Precision Lander, ESA/ESTEC, The Netherlands. jacopo.carradori@esa.int

†GNC Senior Researcher, Department of Guidance, Navigation, and Control, AIAA Senior Member

‡Associate Professor, Section Astrodynamics and Space Mission, AIAA Associate Fellow

technique, known as Meta-Reinforcement Learning (meta-RL) or *learning to learn*, optimizes a meta-policy not only for a single task but for the entire set of tasks from the training distribution. By exploiting prior experience, the network is able to adapt to all scenarios belonging to that distribution, even to those not encountered during the training phase. This feature is particularly useful in uncertain or partially observable environments, where the agent is trained on a subset of experiences sampled from the given uncertainty distributions, to learn a generalized policy. Recurrent Neural Networks (RNNs), such as Long-Short Term Memory (LSTM), have been successfully applied to different spacecraft guidance problems in uncertain scenarios such as interplanetary transfer [10], asteroid hovering [11], and asteroid impactor [12], demonstrating their ability to learn robust policies.

Transformers overcome gradient issues typical of RNNs by employing self-attention mechanisms [13]. This approach captures long-term dependencies by processing data in sequences, and avoiding to compress past information into fixed-size hidden states, as done by LSTMs. Building on Transformers' success in text and image processing, where they enable more efficient dependency extraction and faster training, a variant called Gated Transformer XL (GTrXL) [14] has shown promise in RL applications, particularly for time series and sequential decision-making tasks. For example, Federici and Furfaro [15] demonstrated improved performance of Transformers with respect to fully connected NNs in finding the optimal policy for planetary landing, in presence of several uncertainties, including unmodeled dynamics, imperfect navigation knowledge, and control errors. Transformers significantly enhance robustness by generating control commands based on uncertain vehicle states. During training, the agent experiences various uncertainties and their impacts, learning to handle them correctly. Therefore, this method effectively generates a feedback policy, increasing the robustness and adaptability of the solution.

The main contribution of this paper is the integration of an innovative Transformer-based neural network architecture with meta-RL, to learn a robust guidance and control strategy for the 6-DOF atmospheric powered landing problem. A key advancement compared to existing RL-based GNC studies is that the training phase incorporates accurate environment and vehicle models, including winds, navigation and control errors, uncertainties in initial conditions, and atmospheric and aerodynamic variations, to reflect operational conditions. These additions are important because they ensure a robust policy that adapts to real-world flight conditions, with enhanced adaptability to different scenarios. Furthermore, the GTrXL policy is compared to LSTM results, a type of neural network widely used in similar space guidance applications, to show the performance improvements provided by this innovative method. To illustrate the robustness advantages provided by such a meta-RL method, results are also compared to a conventional G&C solution made of an optimal trajectory tracked by a controller. Using neural network architectures, the agent is trained to learn a robust policy to perform an accurate pinpoint landing, mapping from states to control variables, while satisfying several path and terminal constraints, in an uncertain scenario. The reference vehicle is a reusable first stage, equipped with a throttleable engine, a Thrust Vector Control (TVC) system, and two pairs of aerodynamic fins. The network outputs control commands for the TVC, fins, and thrust while taking mass, translational, and rotational states as inputs. Furthermore, the use of Machine Learning techniques implies that the computational burden lies entirely in the offline network training, while the deployed policy is capable of real-time performance. Hence, in this work, a robust closed-loop policy is obtained, able to run online, in an uncertain, operationally representative environment.

This paper is organized as follows: Sec. II provides an overview of meta-RL. Section III describes the Transformer-based architecture used in this work. Section IV introduces the formulation of the 6-DOF powered landing problem, delineating also the environment and vehicle models used, while Sec. V presents the meta-RL problem and the reward function design. Section VI critically compares the performance of the two meta-RL-based policies, both relative to each other and against an optimal-control-based solution. This analysis examines the strengths and weaknesses of the training processes, focusing on fuel efficiency, robustness, and adaptability to uncertainties. Finally, Sec. VII contains concluding remarks about this work.

II. Meta-Reinforcement Learning

Reinforcement Learning is a branch of Machine Learning, which consists of the most basic idea of learning: the interaction between an agent and an environment produces information about causes and effects, about consequences of actions, and about how to act to achieve a goal [16]. The agent learns to make sequential optimal decisions in a dynamic environment, to maximize the positive feedback it receives. Interactions take place with the environment passing states (or observations) to the agent, which executes actions and receives a reward, based on the actions performed. The environment also determines the transition to new states, which are fed to the agent, so that it can take new actions, repeating the process until termination. The reward is the way to communicate to the agent what to achieve, and it expresses how good certain actions are to achieve a predefined goal. The concept of 'good' and 'bad' is not necessarily

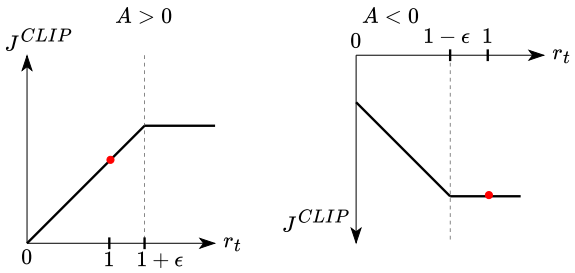


Fig. 1 PPO clipped objective behavior (adapted from [18]).

binary and translating it into an efficient mathematical formulation is one of the most challenging parts of RL. For every training iteration, using its current policy, the agent interacts with the environment, taking actions and receiving a reward at each time step. Based on the goodness of the actions performed in specific states, and the neural network parameters are updated. After several iterations, the optimization terminates yielding the final policy. The goal is to learn an optimal mapping from states to actions.

Early RL tabular methods, stored values for each state-action pair, limiting them to small, discrete problems only. To overcome this, NNs emerged as one of the best function approximators, capable of effectively capturing non-linear relationships. The application of NNs with multiple layers to RL is known as Deep Reinforcement Learning (DRL), enabling solutions to high-dimensional problems previously intractable, due to the computational limitations.

Meta-RL applies Meta-Learning to DRL [17]. The goal of any Meta-Learning technique can be summarized in *learning to learn*. Unlike traditional RL, which optimizes a single policy for a single task, meta-RL trains an agent to succeed in a range of related tasks $\mathbf{T}$, sampled from a continuous distribution $\mathcal{P}(\mathbf{T})$, similarly to how humans leverage prior experience for new, similar tasks. This enables the agent to generalize and adapt its policy efficiently, making it more flexible in new, unseen environments. Each task $\mathbf{T}$ represents a different problem, with its own set of trajectories $D_{\mathbf{T}} = \tau_i \quad (i = 1, ..., n)$ and return $G_{\mathbf{T}}$ (i.e., cumulative sum of rewards). The objective of meta-RL is to train an agent on a limited set of tasks, enabling it to perform optimally in any task within that distribution, even those not encountered during training. A set of network parameters $\boldsymbol{\theta}$ is learned so that $\pi_{\boldsymbol{\theta}}(\mathbf{s}|\mathbf{a}; f_{\boldsymbol{\theta}}(\mathbf{T}))$ is the optimal policy for all tasks $\mathbf{T}$, maximizing the expected cumulative sum of reward over all the different tasks. Hence, the objective function is:

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mathbf{T}\sim\mathcal{P}(\mathbf{T})} \left\{ \mathop{\mathbb{E}}_{\tau\sim\pi(\cdot;f_{\boldsymbol{\theta}}(\mathbf{T}))} [G_{\mathbf{T}}(\tau)] \right\} \tag{1}$$

Meta-RL addresses the challenges that traditional DRL algorithms face when solving multiple related problems with varying environments or observations. Standard fully connected NNs often struggle to specialize outputs for different tasks, typically requiring larger networks, more data, and slower training. Unlike standard NN, meta-RL excels in finding optimal policies for problems that cannot be formulated as Markov Decision Processes (MDPs), such as Partially Observable MDPs, where the optimal control depends on both current and past observations.

The algorithm used here is Proximity Policy Optimization (PPO), which is an on-policy, actor-critic algorithm, based on the concept of conservative improvements between two iterations, to avoid destructive updates [18]. This is actuated by limiting the objective function using a clip operator. The probability ratio r_t of taking a certain action given a certain state, with the current policy with respect to the one at the previous iteration, is:

$$r_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \tag{2}$$

By clipping it with the hyperparameter ϵ, it is forced to stay in an interval around 1. The clipped objective function is:

$$J^{\text{CLIP}}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi_{\boldsymbol{\theta}_k} \\ t=0,...,T}} [\min[r_t(\boldsymbol{\theta}), \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)] A^{\pi_{\boldsymbol{\theta}_k}}(\mathbf{s}_t, \mathbf{a}_t)] \tag{3}$$

By taking the minimum between its first two terms, a conservative (but pessimistic) objective is obtained. Figure 1 shows the behavior of the clipped function. For instance, if the advantage function is positive ($A^{\pi_{\boldsymbol{\theta}_k}} > 0$), it indicates that the chosen action is significantly better than the others. If the probability of taking an action at the current step is

larger than the previous one (r_t>1), the network's parameters are updated to select similar actions in the next iteration. However, the update is constrained to prevent over-exploitation in that direction. Conversely, if the current action gives a negative advantage, the large negative objective value is not clipped, so that the parameters of the network are updated in the opposite direction, to avoid repeating the bad action. Hence, the exploitation of outliers and abruptive changes is avoided, since the policy is updated at every iteration using only an inherently limited set of data.

The objective function can be augmented with two terms. The first term is the quadratic error of the value estimation, which is added only if the policy (actor) and the value (critic) are outputs of the same network. The second one is the entropy term, which can be added to increase the level of exploration. It is reminded that exploration is always included in PPO, given the stochastic policy from which actions are sampled. The complete objective function, where c_1 and c_2 are two parameters to balance value function error and entropy exploration, is:

$$J^{\text{PPO}}(\boldsymbol{\theta}) = J^{\text{CLIP}}(\boldsymbol{\theta}) - c_1(V_{\boldsymbol{\theta}}(\mathbf{s}) - V_{target})^2 + c_2\mathcal{H}(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)) \tag{4}$$

III. Transformer Neural Networks

Transformer NNs offer an efficient alternative to Recurrent NNs for processing time series by using attention layers instead of recurrent layers. This design allows Transformers to process sequences in parallel, unlike RNN, which must handle inputs sequentially. Attention and Transformers, introduced in 2017 in "Attention is all you need" [13], revolutionized generative AI, becoming the foundation of Large Language Models such as ChatGPT, Bard, and Gemini.

Before Transformers, LSTM represented the state of the art for time-series sequence modeling. However, feedback loops and hidden states in RNNs tend to prioritize more recent information, making it difficult to discern dependencies between distant elements in a sequence. *Attention-based* networks eliminate these constraints, allowing parallelization and capturing dependencies over any distance. Additionally, by processing a sequence altogether, they do not compress past information in fixed-size hidden states, as done in LSTM, avoiding suffering from vanishing/exploding gradients.

The attention mechanism calculates similarities between sequence elements to focus on the most relevant parts for a task. Unlike RNNs, which create a single context vector, attention assigns weights to each element (i.e., token) based on their relevance. The Transformer architecture works as follows: a sequence of states is split into several tokens, each representing a time step. For each token, three high-dimensional representations are created: the query, key, and value. They are vectors in a hyperspace, encapsulating the meanings and features of each state. The similarities between keys and queries are computed using the dot product, to generate scores, which are used to update the values, using a weighted sum. In this way, the "meaning" of each token is improved by updating its high-dimensional representation, based on how different tokens relate to each other.

Attention-based networks, with their ability to capture distant relationships, have shown significant performance improvements over LSTMs in many Supervised Learning domains. Their efficiency in handling sequential data makes them ideal for partially observable RL problems, where the trajectory spans over a large number of steps and the crucial observations for a decision are distributed over the entire length of the episode [14]. However, their adoption in RL problems did not take off rapidly, and most of the problems are still solved using LSTMs, due to the easier LSTM implementation and the challenges Transformers face to converge to a meaningful policy. In [14], the authors proposed an evolution of the Transformer architecture, suited to be used in an RL framework, called Gated Transformer XL.

Its structure is similar to that of a standard Transformer, as shown in Fig. 2b. The two main elements are the masked Multi-Head Attention and the Position-Wise Feed Forward network. However, the normalization is now placed before each sub-layer, and a gating mechanism, is located immediately after each sub-layer, replacing the residual connection.

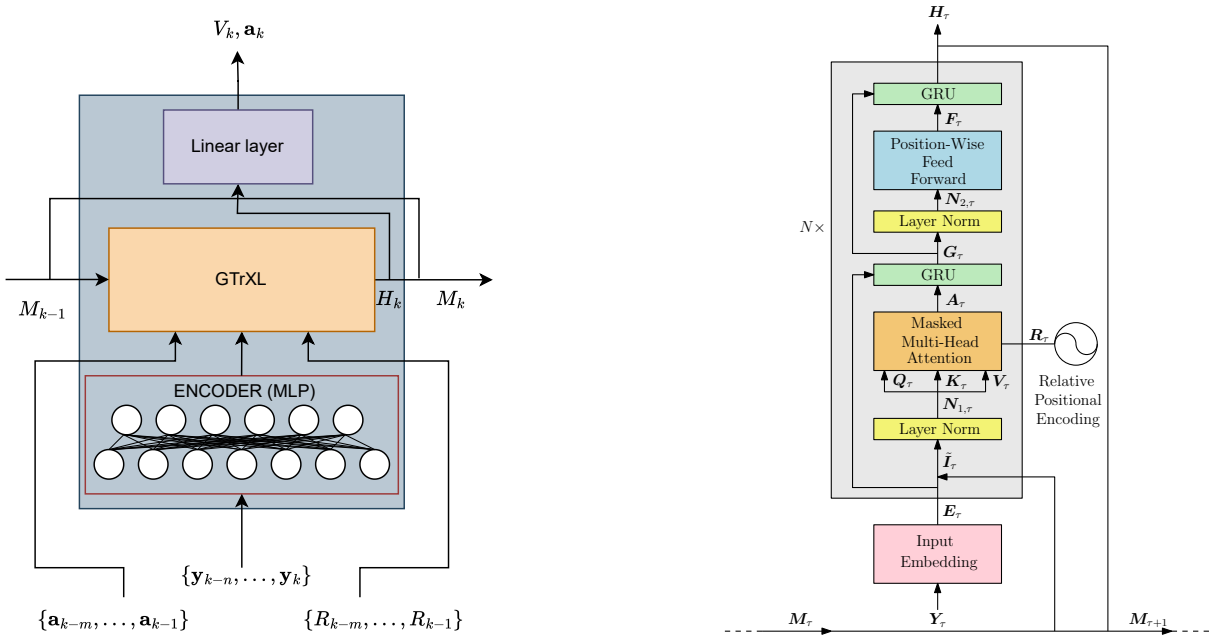
GTrXL processes an input $\mathbf{Y}$ of total length L, which is a sequence of n temporally consecutive observations (each with dimension m). It is first divided into τ segments of length n, where each of the τ-th segments is:

$$\mathbf{Y}_\tau = [\mathbf{y}_{\tau,1}, \ldots, \mathbf{y}_{\tau,n}] \in \mathbb{R}^{n\times m}, \quad \tau = 1, \ldots, L/n \tag{5}$$

For the first Transformer layer ($k = 1$), the input $\mathbf{Y}_\tau$ passes through the initial encoder of Multi-Layer Perceptron, becoming $\mathbf{E}_\tau$, which is a high-dimensional representation of the input states sequence. $\mathbf{E}_\tau$ can be augmented by adding a set of actions and rewards from the previous m time steps. Moreover, the input to the Transformer layer is augmented by a memory $\mathbf{M}^k_\tau$, which contains the outputs $\mathbf{H}^k_\tau$ of that specific k-th Transformer layer from the previous T segments:

$$\mathbf{M}^k_\tau = \left[\mathbf{H}^k_{\tau-T}, \mathbf{H}^k_{\tau-T+1}, \ldots, \mathbf{H}^k_{\tau-1}\right] \tag{6}$$

Next, the input passes through the normalization layer. After that, the queries $\mathbf{Q}^k_\tau$, keys $\mathbf{K}^k_\tau$, and values $\mathbf{V}^k_\tau$ are generated and fed to the multi-head attention. For each attention head, the attention function is computed simultaneously,



(a) Schematic of GTrXL-based neural network.

(b) Schematic of GTrXL module, from [15].

Fig. 2 GTrXL network architecture.

using the scaled masked dot-product, on a set of queries, keys, and values packed into matrices **Q**, **K**, and **V**. The dot-product between queries and keys represents how similar/coincident are two tokens representations, producing the scores for each pair. These scores are scaled with the softmax function to generate weights between 0 and 1, ensuring proper normalization before performing the weighted sum of the values. By multiplying the attention scores and the values, only those values multiplied by high scores are kept, filtering out irrelevant elements with poor affinity. The masked operator (χ) ensures causality in temporal sequences by preventing the use of future states in time.

$$\mathbf{A}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\chi(\mathbf{Q}\mathbf{K}^T)}{\sqrt{d_k}}\right)\mathbf{V} \tag{7}$$

The concatenated outputs of all the attention heads are "delta vectors" representing relationships between tokens, to be added to the embeddings to update their high-dimensional representations. The multi-head attention output $\mathbf{A}_\tau^k$ is first passed through a ReLU, then through the gating mechanism, together with the current input of the Transformer layer:

$$\mathbf{G}_\tau^k = \mathbf{g}(\mathbf{I}_\tau^k, \max(0, \mathbf{A}_\tau^k)) \tag{8}$$

The three final steps to get the output $\mathbf{H}_\tau^k$ are: a second normalization layer, a position-wise fully connected network, and a gating mechanism. In [14], the authors found out that GTrXL achieved stable, fast, and reliable training. In challenging, high-dimensional continuous environments, GTrXL consistently outperformed LSTM in terms of policy performance and robustness to hyperparameter sensitivity.

This improvement is attributed to the placement of the normalization layer only on the input stream of the sub-layers, allowing an identity mapping from inputs to outputs. This helps during early training phases when the sub-layers output near-zero values, providing untransformed observations to the policy and value outputs. This approach facilitates learning a Markovian policy first, which is then refined with the use of memory to adapt to multiple tasks [14]. In standard Transformers, input sequences are divided into shorter segments and processed one at a time. In contrast, GTrXL reuses its output from previous segments as a form of memory, similar to the feedback mechanism in RNNs.

Considering an architecture as in Fig. 2a, the encoder input is composed of the last n observations $\mathbf{y}_t$ (from timesteps $t-n$ to t). Thus, the GTrXL block takes the encoder output, the memory $\mathbf{M}_t$ composed of the last T outputs of the Transformer layer, and the last $m-1$ actions and rewards as augmented input. The output of GTrXL $\mathbf{H}_t$ passes through

a linear layer to obtain the current actions $\mathbf{a}_t$ and value function V_t. Recursively, the previous output of the Transformer $\mathbf{H}_{t-1}$ depends on the preceding n time steps and the previous memory vector $\mathbf{M}_{t-1}$. Hence, each time step's output relies on the most recent n steps of the most recent trajectory and the T vectors from the last T GTrXL outputs, containing information about the last T sequences of length n, from current and previous trajectories.

IV. 6-DOF Powered Landing Problem

The space vehicle is assumed to be a rigid body with variable mass, neglecting sloshing and bending effects. The Equations of Motion (EOMs) are expressed according to Classical Mechanics, expressed in the **UEN** (up-east-north) reference frame, which is a rotating frame fixed on the landing site position. Hence, to ensure the validity of Newton's laws, fictitious forces have to be introduced. Thus, the translational motion EOMs in **UEN** frame are:

$$\begin{aligned}\dot{\mathbf{r}}^{\mathbf{UEN}} &= \mathbf{v}^{\mathbf{UEN}} \\ \dot{\mathbf{v}}^{\mathbf{UEN}} &= \mathbf{g}^{\mathbf{UEN}} + \frac{1}{m}(\mathbf{F}^{\mathbf{UEN}}_{thr} + \mathbf{F}^{\mathbf{UEN}}_{aero,body} + \mathbf{F}^{\mathbf{UEN}}_{aero,fins}) - 2\boldsymbol{\omega} \times \mathbf{v}^{\mathbf{UEN}} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}^{\mathbf{UEN}})\end{aligned} \tag{9}$$

$\mathbf{r}^{\mathbf{UEN}}$ and $\mathbf{v}^{\mathbf{UEN}}$ represent the position and velocity of the vehicle's Center of Mass (CoM) in the landing site reference frame. The terms $\mathbf{g}^{\mathbf{UEN}}$, $\mathbf{F}^{\mathbf{UEN}}_{thr}$, $\mathbf{F}^{\mathbf{UEN}}_{aero,body}$, and $\mathbf{F}^{\mathbf{UEN}}_{aero,fins}$ represent the gravitational acceleration, the propulsive force, and the aerodynamic force of body and fins, expressed in landing site reference frame. The terms $2\boldsymbol{\omega} \times \mathbf{v}^{\mathbf{UEN}}$ and $\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}^{\mathbf{UEN}})$ represent the apparent accelerations due to the vehicle's motion in the rotating frame: Coriolis and centrifugal acceleration terms, respectively. Earth's rotation rate is taken equal to $\boldsymbol{\omega} = 7.292115 \times 10^{-5}$ rad/s.

The rotational dynamics of the vehicle body is defined by the Euler equations:

$$\dot{\boldsymbol{\omega}}^{\mathcal{B}} = \mathbf{J}^{-1}_{\mathcal{B}}[\mathbf{M}^{\mathcal{B}}_{thr} + \mathbf{M}^{\mathcal{B}}_{aero,body} + \mathbf{M}^{\mathcal{B}}_{aero,fins} - \boldsymbol{\omega}^{\mathcal{B}} \times \mathbf{J}_{\mathcal{B}}\boldsymbol{\omega}^{\mathcal{B}}] \tag{10}$$

$\boldsymbol{\omega}^{\mathcal{B}}$ represents the rotation rates of the vehicle expressed in the body axes with respect to the **UEN** landing site reference frame. $\mathbf{J}_{\mathcal{B}}$ is the inertia matrix of the vehicle expressed in the body frame, while $\mathbf{M}^{\mathcal{B}}_{thr}$, $\mathbf{M}^{\mathcal{B}}_{aero,body}$, $\mathbf{M}^{\mathcal{B}}_{aero,fins}$ are the propulsive moments and aerodynamic moments of body and fins, expressed in body axes.

The vehicle's body axes attitude is propagated using the quaternion kinematics equation:

$$\dot{\mathbf{q}}^{\mathcal{B}} = \frac{1}{2}\Omega \cdot \boldsymbol{\omega}^{\mathcal{B}} = \frac{1}{2}\begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix}\begin{pmatrix}\omega_1 \\ \omega_2 \\ \omega_3\end{pmatrix} \tag{11}$$

Finally, since the vehicle mass m varies throughout the flight, its rate of change is propagated as additional EOM:

$$\dot{m} = \frac{T}{I_{sp} g_0} \tag{12}$$

T is the thrust magnitude, I_{sp} and g_0 are the specific impulse and the gravitational acceleration constant at sea level.

A. External forces

The external forces acting on the rocket during the landing phase come from three different sources: gravity, propulsion, and aerodynamics.

The gravitational force is the force exerted by the central body on the vehicle. Expressed in Cartesian coordinates centered at the landing site frame (**UEN**), it can be approximated as:

$$\mathbf{F}^{\mathbf{UEN}}_{\mathbf{G}} = m\mathbf{g}^{\mathbf{UEN}} = -m\,[g, \quad 0, \quad 0]^T \tag{13}$$

Given a propulsion frame aligned with the thrust force vector, the thrust force in the body frame is expressed as in [19]:

$$\mathbf{F}^{\mathcal{B}}_{thr} = [T\cos\epsilon_T \cos\psi_T, \quad T\cos\epsilon_T \sin\psi_T, \quad -T\sin\epsilon_T]^T \tag{14}$$

ϵ_T, ψ_T are the thrust force angles, due to TVC actuators, with respect to the body frame.

The aerodynamic forces are relevant during the landing problem, affecting the motion of the vehicle. They depend on the vehicle shape, on its orientation with respect to the velocity vector (aerodynamic angles α and β), and on the dynamic pressure $\bar{q}$.

In this work, the coefficients are stored in lookup tables, already in the body frame, as axial (C_A) and normal coefficients (C_N). They are then transformed to C_X, C_Y, and C_Z, using the aerodynamic angles α and β. Therefore, the aerodynamic forces X, Y, and Z, in the body reference frame, are expressed as:

$$\mathbf{F}^{\mathcal{B}}_{aero,body} = [X, \quad Y, \quad Z]^T = \bar{q} S_{ref} [C_X, \quad C_Y, \quad C_Z]^T \tag{15}$$

In addition to the aerodynamic forces generated by the body of the rocket, the aerodynamic forces generated by the fins are also included. Given the reduced fin area with respect to the vehicle body area, the axial force of the fins is neglected, while only the component normal to the fin surface is considered [20]. Neglecting flow separation, the normal contribution has a sinusoidal dependence on the fin local angle of attack (α_{local}). $\bar{C}_{fin}$ is the maximum normal fin force coefficient. Thus, the normal fin force coefficient in the fin reference frame is:

$$C_{fin}(\alpha_{local}) = \bar{C}_{fin} \sin(\alpha_{local}) \tag{16}$$

The local angle of attack is computed differently for the two pairs of fin. Since one pair is associated with the control of the pitch plane (fins 1 and 2) and one with the control of the yaw plane (fins 3 and 4), they are respectively related to angle of attack and angle of sideslip:

$$\begin{aligned} \alpha_{local,i} &= \beta_{fin,i} - \alpha \qquad i = 1,2 \\ \alpha_{local,i} &= \beta_{fin,i} - \beta \qquad i = 3,4 \end{aligned} \tag{17}$$

The aerodynamic force normal to the fin surface is then calculated, in each fin frame ($\mathcal{F}, x$), as:

$$F^{\mathcal{F},x}_{aero,Y_{fin,i}} = \bar{q} S_{fin} C_{fin}(\alpha_{local}) \tag{18}$$

Then, for each aerodynamic force of the four fins, a rotation takes place, to express them in the body frame. The fins controlling the pitch plane give force components along the $X_{\mathcal{B}}$ and $Z_{\mathcal{B}}$ axes:

$$\mathbf{F}^{\mathcal{B}}_{aero,fin,i} = \begin{pmatrix} F^{\mathcal{F},pitch}_{aero,Y_{fin,i}} \sin(-\beta_{fin,i}) \\ 0 \\ F^{\mathcal{F},pitch}_{aero,Y_{fin,i}} \cos(-\beta_{fin,i}) \end{pmatrix} = \begin{bmatrix} \cos(-\beta_{fin,i}) & -\sin(-\beta_{fin,i}) & 0 \\ 0 & 0 & -1 \\ \sin(-\beta_{fin,i}) & \cos(-\beta_{fin,i}) & 0 \end{bmatrix} \begin{pmatrix} 0 \\ F^{\mathcal{F},pitch}_{aero,Y_{fin,i}} \\ 0 \end{pmatrix} \qquad i = 1,2 \tag{19}$$

The fins controlling the yaw plane give force components along the $X_{\mathcal{B}}$ and $Y_{\mathcal{B}}$ axes:

$$\mathbf{F}^{\mathcal{B}}_{fin,i} = \begin{pmatrix} -F^{\mathcal{F},yaw}_{aero,Y_{fin,i}} \sin(-\beta_{fin,i}) \\ F^{\mathcal{F},yaw}_{aero,Y_{fin,i}} \cos(-\beta_{fin,i}) \\ 0 \end{pmatrix} = \begin{bmatrix} \cos(-\beta_{fin,i}) & -\sin(-\beta_{fin,i}) & 0 \\ \sin(-\beta_{fin,i}) & \cos(-\beta_{fin,i}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} 0 \\ F^{\mathcal{F},yaw}_{aero,Y_{fin,i}} \\ 0 \end{pmatrix} \qquad i = 3,4 \tag{20}$$

B. External Moments

The external moments acting on the vehicle are the propulsive and aerodynamic moments, generated by the body of the vehicle and its fins. Given the relatively small size, the gravity gradient over the vehicle is negligible. Thus, the gravitational moments are not considered in the problem.

The thrust moments $\mathbf{M}^{\mathcal{B}}_{thr}$ acting on the vehicle in the body frame are computed with the cross product between the propulsion force $\mathbf{F}^{\mathcal{B}}_{thr}$ and fixed engine gimbal position relative to the CoM ($\mathbf{x_T} - \mathbf{x}_{CoM}$).

It is assumed that the aerodynamic moments $\mathbf{M}^{\mathcal{B}}_{aero,body}$ generated by the body of the vehicle around its CoM are caused only by the aerodynamic forces $\mathbf{F}^{\mathcal{B}}_{aero,body}$ applied in the center of pressure (CP), calculated as the cross product between the aerodynamic forces and the distance between CoM and CP ($\mathbf{x}_{CP} - \mathbf{x}_{CoM}$). Additional moment damping coefficients are very difficult to estimate for rockets and therefore they are assumed equal to zero in this work. As pointed out in [21], it is a conservative assumption since the damping term would make the response slower and more stable. The center of pressure position $\mathbf{x}_{CP}$ depends on the aerodynamic angles and Mach number and it is tabulated together with the aerodynamic coefficients.

Similarly to the body, also the fins' aerodynamic moments $\mathbf{M}^{\mathcal{B}}_{aero,fins}$ are calculated only using the fins aerodynamic forces $\mathbf{F}^{\mathcal{B}}_{aero,fin,i}$ and the distance between each of their CP and the vehicle CoM ($\mathbf{x}_{finCP,i} - \mathbf{x}_{CoM}$). Given the small size, each fin's CP is assumed to be the fixed attachment point on the rocket.

Table 1 Initial conditions in UEN landing site reference frame, and uncertainties.

Description	Symbol	Unit	Nominal	Dispersion
Elevation position	x_0	m	2000	$\mathcal{U}$(1900,2100)
East position	y_0	m	200	$\mathcal{U}$(180,220)
North position	z_0	m	-200	$\mathcal{U}$(-220,-180)
Vertical velocity	v_{x_0}	m/s	-100	$\mathcal{U}$(-110,-90)
Horizontal velocity (East)	v_{y_0}	m/s	-25	$\mathcal{U}$(-30,-20)
Horizontal velocity (North)	v_{z_0}	m/s	25	$\mathcal{U}$(20,30)
Roll	ϕ_0	deg	0	$\mathcal{U}$(-5,5)
Pitch	θ_0	deg	20	$\mathcal{U}$(15,25)
Yaw	ψ_0	deg	20	$\mathcal{U}$(15,25)
Roll rate	$\omega_{x,0}$	deg/s	0	$\mathcal{U}$(0,0)
Pitch rate	$\omega_{y,0}$	deg/s	-5	$\mathcal{U}$(-7.5,-2.5)
Yaw rate	$\omega_{z,0}$	deg/s	-5	$\mathcal{U}$(-7.5,-2.5)
Mass	m_0	kg	4000	$\mathcal{U}$(3900, 4100)
Density uncertainty	$\Delta\rho$	kg/m^3	0	$\mathcal{U}$(-10%,10%)
Aerodynamic uncertainty	ΔC_X,ΔC_Y,ΔC_Z	-	0	$\mathcal{U}$(-15%,15%)

C. Initial conditions and uncertainties

The mission scenario in this work is the terminal phase of a reusable launch vehicle re-entering Earth's atmosphere, focusing on the powered landing phase after the aerodynamic descent. This phase is the final stage in both Down-Range and Return-To-Launch-Site landings, and it is largely independent of the mission profile. The nominal initial conditions, and the corresponding dispersions, are shown in Table 1.

The initial altitude is set at 2000 m, while the horizontal initial position is chosen to be 10% of the initial altitude, following the scenario adopted in recent CALLISTO studies [6, 22]. Many landing videos of Falcon 9 missions show the start of the propulsive phase at about 2 km. The initial vertical velocity is 100 m/s, scaled to be 20 times smaller than the initial altitude, as done in [15]. Initial horizontal velocities, pitch and yaw angles, and angular rates are chosen with engineering reasoning to reflect real flight conditions. The nominal mass value is the same used by [23] and [19].

Table 1 presents the initial dispersions that are sampled from uniform distributions at the start of each simulation. 10% uncertainty is chosen for the horizontal position, while only 5% for the initial altitude since it is known with more precision because it is the variable that triggers the beginning of the propulsive landing. Dispersion on vertical velocity is 10%, similar to what is done by [15], while it is 20% on horizontal components, due to lower absolute value. Dispersions on rotational states are chosen to be a few degrees (or degrees per second) since these variables are not very large in absolute terms. Mass dispersion is 2.5%, close to the value used by [24].

The most uncertain dynamic variables are those related to atmospheric and aerodynamic properties. At such low altitudes, density and aerodynamic coefficients can have large local variations due to changes in weather conditions and imperfect knowledge. At the start of each simulation, values are sampled from the uniform distributions shown in Table 1, similar to the ones used in [25]. They are kept constant for the entire simulation and applied as multiplicative factors:

$$\rho = (1 + \Delta\rho)\rho_{nom}; \quad C_i = (1 + \Delta C_i)C_{i_{nom}} \tag{21}$$

D. Control and navigation errors

Variations of initial conditions and imperfect knowledge of complex dynamics are great sources of uncertainties that affect the vehicle's flight. However, other error sources affect guidance and control algorithms, primarily related to hardware, such as the errors introduced by the navigation and control modules.

The navigation module introduces errors due to biases, misalignment, and intrinsic sensor inaccuracies. Hence, to simulate navigation errors, Gaussian noise is added to the actual states $\mathbf{x}$, generating the estimated states $\bar{\mathbf{x}}$:

$$\bar{\mathbf{x}} = \mathbf{x} + \mathcal{N}(0, \sigma_{\mathbf{x}}) \tag{22}$$

Table 2 Navigation uncertainties.

State variable	1σ
σ_{r_x},σ_{r_y},σ_{r_z}	0.15 m
σ_{v_x},σ_{v_y},σ_{v_z}	0.1 m/s
σ_{ω_x},σ_{ω_y},σ_{ω_z}	0.15 deg/s
σ_ϕ, σ_θ, σ_ψ	0.15 deg

Table 3 Control uncertainties.

Control variable	1σ
$\sigma_{T_{bias}}$	1%
$\sigma_{T_{error}}$	1%
$\sigma_{\dot{\epsilon}_T}$, $\sigma_{\dot{\psi}_T}$	0.25 deg/s
$\sigma_{\dot{\beta}_{fins,pitch_1}}$, $\sigma_{\dot{\beta}_{fins,pitch_2}}$	0.25 deg/s
$\sigma_{\dot{\beta}_{fins,yaw_3}}$, $\sigma_{\dot{\beta}_{fins,yaw_4}}$	0.25 deg/s

In this way, errors are sampled from pre-defined distributions at each time step, and added to the actual states, to avoid accumulation. The estimated states are used by the GNC system to calculate the control actions, while the actual states to propagate the dynamics. Table 2 shows the standard deviation for each error, which are assumed uncorrelated with each other.

Errors in the control module arise because imperfect actuators' performance deviates from commanded values. Moreover, actuator dynamics are not considered, missing the modeling of transients due to friction, inertia, and damping effects. To model these real-world effects, errors are added to the commanded control variables output by the neural network. TVC and fin deflection rate errors are sampled from Gaussian distributions at each time step, and summed to the commanded values. Thrust magnitude error combines two terms. A constant bias, sampled from a Gaussian distribution at the start of each episode, multiplies the nominal thrust value along the entire trajectory. An additional multiplicative factor is added, sampled from a Gaussian distribution at each time step, to simulate uncertain thrust values. Table 3 presents the standard deviation for each error. They are assumed uncorrelated with each other.

E. Environment models

Given the short flight time and the relatively low altitude, it is chosen to approximate the gravitational acceleration as a simple central-body field, calculated as an inverse quadratic function of the radial distance from the center of the Earth.

The atmospheric model used in this work is US76 [26], as a result of a trade-off between modeling accuracy and computational time. It calculates temperature, pressure, and density only as functions of altitude, divided into multiple layers. The atmosphere is assumed a perfect ideal gas, homogeneously mixed, and in hydrostatic equilibrium.

The wind acts as a perturbation on the vehicle, altering its aerodynamic angles, deviating its motion, and increasing the dynamic loads. It is modeled as the sum of a steady component, and an unpredictable, short-duration component.

The steady wind component is generated using the Horizontal Wind Model 14 (HWM14) [27], which is an empirical model that provides the meridional w_{NS} (north-south) and zonal w_{EW} (east-west) wind velocity components based on location, altitude, and time of year, including solar disturbances. The model assumes negligible vertical wind due to its much smaller magnitude compared to horizontal winds.

Given the relatively small altitude range and short time frame, only small-scale perturbations are considered. A first-order auto-regressive model is used in this simulator, computing a perturbation at each position from the correlated perturbation value at the previous position. Considering a normalized variate $\mu(\mathbf{r}_k)$ (where $\mu(\mathbf{r}_k)$ is the value of the deviation of the variable with respect to the mean, at the position $\mathbf{r}_k$, divided by the standard deviation), the perturbation at the following position $\mathbf{r}_{k+1}$ is calculated as:

$$\mu(\mathbf{r}_{k+1}) = p\mu(\mathbf{r}_k) + \sqrt{1-p^2}q(\mathbf{r}) \tag{23}$$

$q(\mathbf{r})$ is a Gaussian-distributed random number with a mean equal to 0 and standard deviation equal to 1, while p is the auto-correlation term between values at two successive positions $\mathbf{r}_k$ and $\mathbf{r}_{k+1}$. In this work, given the short time scale and horizontal displacement, the auto-correlation factor is assumed only dependent on the vertical position:

$$p(\delta\mathbf{r}) = exp\left(-\frac{\delta r_x}{L_z}\right) \tag{24}$$

Wind gusts are added on top of this. They are modeled as a "1-cosine" shape, so that there is a peak at a given altitude, and it is dumped out at altitudes above and below it. For each new simulation, there is a 50% chance of creating a wind gust in the meridional or zonal direction. As shown in [28], the equation used to model them is the following:

$$\mathbf{v}_{wind,gust} = \begin{cases} 0 & \text{if } h < 0 \text{ or if } h > 2d \\ \frac{\mathbf{v}_{max}}{2}\left[1 - \cos\left(\frac{\pi h}{d}\right)\right] & \text{if } 0 \le h \le 2d \end{cases} \tag{25}$$

d is the gust half-width, randomly varying between 125 m and 375 m. $\mathbf{v}_{max}$ is the gust maximum magnitude vector in correspondence of the half-width, randomly varying between ±5 m/s and h the position of the vehicle inside the gust width. The wind gust is superimposed on the sum of mean wind and turbulence. Thus, the total wind is computed as:

$$\mathbf{v}_{wind}(\mathbf{r}) = \begin{pmatrix} 0 \\ w_{EW}(\mathbf{r}) + \mu(\mathbf{r})\sigma_{w_{EW}}(\mathbf{r}) \\ w_{NS}(\mathbf{r}) + \mu(\mathbf{r})\sigma_{w_{NS}}(\mathbf{r}) \end{pmatrix} + \mathbf{v}_{wind,gust} \tag{26}$$

F. Vehicle model

The reference vehicle is a first stage rocket booster, similar to real-world current and future reusable launch vehicles, such as Falcon 9 and CALLISTO [5, 6]. It is equipped with two sets of aerodynamic fins, and with a single engine, gimballed around two axes. It has an axis-symmetric shape, a diameter of 1.5 m, and a total length of 15 m. The vehicle parameters are summarized in Table 4. They are either taken from [19], [23], or chosen using engineering reasoning.

Modeling the Mass, Center of Mass, and Inertia (MCI) with enough accuracy is crucial to have realistic simulations. For instance, CoM position affects the moment arm in external moment calculations, while inertia influences rotational dynamics. A low-fidelity model with a constant CoM can lead to significant discrepancies, as actuator deflections calculated for a fixed CoM may generate control moments not matching real-flight conditions, due to a different lever arm. Therefore, a more accurate analytical model is used for the MCI, and its parameters are presented in Table 4. The CoM is assumed to lie on the rocket's longitudinal axis and the inertia matrix is considered diagonal, containing only the principal moments of inertia. Due to the vehicle's axisymmetry, two of them are equal:

$$\mathbf{J}(t) = \begin{bmatrix} J_A(t) & 0 & 0 \\ 0 & J_N(t) & 0 \\ 0 & 0 & J_N(t) \end{bmatrix} \tag{27}$$

As described in [19], the MCI model is based on the fuel mass depletion, with respect to the initial value:

$$\eta(t) = \frac{m_{prop}(t)}{m_{prop}(t_0)} \tag{28}$$

m_{prop} is the total propellant mass, which is the sum of fuel m_{fuel} and oxidized m_{ox}.

The CoM is calculated using the varying masses and height levels of fuel and oxidizer in the tanks:

$$x_{CoM}(t) = \frac{1}{m(t)} \left[m_{fuel}(t) \left(h_{tank,fuel} + \frac{h_{fuel}(t)}{2} \right) + m_{ox}(t) \left(h_{tank,ox} + \frac{h_{ox}(t)}{2} \right) + m_{dry} h_{dry} \right] \tag{29}$$

$h_{fuel}(t)$ and $h_{ox}(t)$ is the level of propellant in each tank at a given time step, calculated as:

$$h_x(t) = \eta(t) d_{tank,x} \tag{30}$$

Given constant propellant densities, $\frac{h_x(t)}{2}$ represents the CoM height of each propellant with respect to the tank's base.

The axial J_A and lateral components J_N are calculated as:

$$J_A(t) = \frac{1}{2} m_{prop}(t) r_{tank}^2 + J_{A,dry} \tag{31}$$

$$\begin{aligned} J_N(t) = & \frac{1}{12} m_{fuel}(t) \left(3r_{tank}^2 + h_{fuel}^2(t) \right) + \frac{1}{12} m_{ox}(t) \left(3r_{tank}^2 + h_{ox}^2(t) \right) + J_{N,dry} + \\ & m_{fuel}(t) \left(h_{tank,fuel} + \frac{h_{fuel}(t)}{2} - x_{CoM}(t) \right)^2 + m_{ox}(t) \left(h_{tank,ox} + \frac{h_{ox}(x)}{2} - x_{CoM}(t) \right)^2 + \\ & m_{dry} (h_{dry} - x_{CoM})^2 \end{aligned} \tag{32}$$

Table 4 Vehicle parameters.

Description	Symbol	Value	Unit
Initial wet mass	m_0	4000	kg
Dry mass	m_{dry}	2750	kg
Minimum thrust	T_{min}	40	kN
Maximum thrust	T_{max}	100	kN
Thrust rate limit	$\dot{T}_{max}$	10	kN/s
Specific impulse	I_{sp}	282	s
Maximum TVC deflection angle	$\epsilon_{T_{max}}, \psi_{T_{max}}$	10	deg
Maximum fin deflection angle	$\beta_{fin,i_{max}}$	25	deg
Maximum TVC deflection rate	$\dot{\epsilon}_{T_{max}}, \dot{\psi}_{T_{max}}$	15	deg/s
Maximum fin deflection rate	$\dot{\beta}_{fin,i_{max}}$	200	deg/s
Fin area	S_{fin}	0.54	m^2
Longitudinal fin position wrt bottom stage	X_{fins}	11.1	m
Engine gimbal position wrt bottom stage	$\mathbf{x}_T$	(0.3, 0, 0)	m
Base fuel tank location wrt to bottom stage	$h_{tank,fuel}$	1.2	m
Base oxygen tank location wrt to bottom stage	$h_{tank,ox}$	5.4	m
Fuel tank length	$d_{tank,fuel}$	3.3	m
Oxygen tank length	$d_{tank,ox}$	5.97	m
Tank radius	r_{tank}	0.7	m
Dry rocket CoM position	h_{dry}	3.585	m
Dry rocket axial moment of inertia	$J_{A,dry}$	5880	kg m^2
Dry rocket lateral moment of inertia	$J_{N,dry}$	39000	kg m^2

V. Meta-RL reward function design

The guidance system employs meta-RL to determine the optimal closed-loop guidance strategy for landing a rocket. This method allows the agent to use its past experiences to select the best control action at each time step.

The control vector output of the policy is composed of 7 elements: thrust rate, two TVC deflection rates, and four fin deflection rates:

$$\mathbf{u} = \left[\dot{T}, \dot{\epsilon}_T, \dot{\psi}_T, \dot{\beta}_{fins,pitch_1}, \dot{\beta}_{fins,pitch_2}, \dot{\beta}_{fins,yaw_3}, \dot{\beta}_{fins,yaw_4}\right] \tag{33}$$

The choice of the rates instead of the deflection angles is driven by the necessity of enforcing constraints on both the rates and the deflection angles. The same reasoning applies to thrust magnitude and thrust rate.

The inputs to the neural network are all the observations $\mathbf{x}$ available from the dynamics. If the states are not perturbed, the observations are coincident with the states.

$$\begin{aligned}\mathbf{x} = [&r_x, r_y, r_z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, q_1, q_2, q_3, q_4,\\ &\epsilon_T, \psi_T, \beta_{fins,pitch_1}, \beta_{fins,pitch_2}, \beta_{fins,yaw_3}, \beta_{fins,yaw_4}, T, m, t]\end{aligned} \tag{34}$$

Using `pyrlprob`, an open-source Python library that facilitates the training, evaluation, and post-processing of `Gym`-based environments using `Ray-RLlib`, a framework is created to define the neural network and the rocket landing dynamics needed for the training and evaluation processes.

To ensure consistent magnitudes across state variables, the entire problem is normalized, selecting a minimal set of reference units. The normalization facilitates the NN training, which is improved if it handles values close to the unit. Similarly to what is done by [5], the chosen reference variables are: nominal initial altitude (2000 m), nominal initial mass (4000 kg), gravitational acceleration (9.80665 m/s), ambient temperature (288.15 K) and pressure (101325 Pa). Manipulating the first three variables, reference values for time, velocity, and acceleration are found. Finally, each state variable, vehicle, or environment parameter is divided by the reference value corresponding to its unit of measure.

Training a neural network to learn a policy involves running numerous episodes over several iterations. In each iteration, new simulations are initialized with uncertainties and initial conditions sampled from given distributions. During each episode, actions are performed based on the current network policy, and the EOMs are propagated. Initially, the agent's policy is random, with unrealistic behavior, such as the vehicle rotating with extremely unrealistic angular rates and attitude. As training progresses, the agent learns to identify good and bad actions based on its current state, refining the neural network's weights and biases to improve its policy over time.

One of the most crucial and challenging aspects of RL is defining the reward function. This scalar value communicates to the agent what needs to be achieved, indicating how good a specific action is at a given state in pursuit of the desired goal. The reward function is central to ensuring that the network learns the correct policy. A poorly designed reward function can lead the policy to diverge completely from the intended objective. Since the NN is trained to maximize the expected reward, if the reward function does not align with the user's goal, the agent will still learn a policy, but it may be entirely different from what was intended.

The starting point typically involves understanding the underlying physical problem to identify its key features and defining a clear goal that represents the policy the network should learn, similar to an objective function in optimization. Reward functions can be sparse, providing bonuses and penalties only at the end of each episode, or dense, offering rewards at each time step. While sparse rewards are often more optimal, they can lead to slow or ineffective training. In fact, the network may struggle to capture relationships between a single terminal reward and the entire sequence of states and actions throughout the trajectory, especially in problems with many time steps, states, and controls, where it would condense a lot of information of the entire episode. For instance, it is hard for the agent to understand the cause of a poor trajectory from a single scalar reward in the early stages of training, when the policy is random, and the rocket exhibits erratic behavior, such as rotating rapidly along each axis or landing far from the site. In complex dynamics problems with many states, controls, and long episodes, it is often preferred to incorporate bonuses and penalties at each step to guide the agent toward the desired terminal states.

Providing dense rewards at each time step helps to guide the vehicle toward learning the correct policy. However, this approach can slightly reduce optimality because the trajectory becomes more "constrained" by the shape of the reward function. Logarithmic and exponential functions are useful in guiding the vehicle towards "attractive points" [29] due to their steep changes near these points. Additionally, quadratic functions can accelerate convergence.

In the 6-DOF rocket landing problem of this work, the primary goal is to land the vehicle with specific terminal properties, such as proximity to the landing site, minimal vertical and horizontal velocities, a controlled vertical angle, and acceptable residual angular velocity. Minimal fuel consumption is also a key factor. Therefore, a reward function, made of a component (*shaping reward*) at each time step and a term (*terminal reward*) at the episode's end, is designed to guide the agent in learning the optimal policy while adhering to path and terminal constraints:

- **Drive position components toward landing site**: terminal horizontal position < 10 m radius
- **Guide velocity to zero at landing site**: terminal vertical velocity < 2 m/s, terminal horizontal velocity < 1.5 m/s
- **Avoid large angular rates**: angular rates < 15 deg/s for each axis during flight, terminal angular rates < 3 deg/s for each axis at touchdown
- **Keep the vehicle in a stable attitude**: terminal vertical angle < 3 deg
- **Minimize mass consumption**

In the next paragraphs, each term of the reward function is described.

A. Position

A logarithmic dense reward function is defined for the position component, giving a bonus that increases as the rocket approaches the landing site. The reward is zero at each episode's initial position and theoretically infinite at the exact landing point $\mathbf{r}=\mathbf{0}_3$. This bonus is applied only if the rocket is descending, while, if it is hovering or ascending, a fixed penalty is given to discourage this behavior.

$$R_{pos} = \begin{cases} -0.015 \left\{ \text{sign}\left[\log\left(\frac{\left\| \frac{r_x}{r_{x0}}, \frac{r_y}{r_{y0}}, \frac{r_z}{r_{z0}} \right\|}{\sqrt{3}} \right) \right] \log\left(\frac{\left\| \frac{r_x}{r_{x0}}, \frac{r_y}{r_{y0}}, \frac{r_z}{r_{z0}} \right\|}{\sqrt{3}} \right)^2 \right\}, & \text{if } r_x(t) \leq r_x(t-1) \\ -0.2, & \text{otherwise} \end{cases} \tag{35}$$

The logarithm's argument is the norm of the position components, each normalized by its initial value and divided by the square root of three. This setup ensures that only positions within the initial conditions yield a positive bonus, with the argument being one at the initial position and zero at the target position. To enhance convergence toward the

"attractive" region near the landing site, the square of this logarithm is used. However, squaring the expression could result in a positive bonus for positions higher or farther than the initial conditions. To address this, the expression is multiplied by its sign, ensuring positive rewards only within the initial boundaries and negative penalties otherwise.

This expression ensures that the region near the landing site offers significant rewards, driving the agent toward it. Thanks to the collected experience, the agent learns the desired policy by associating large bonuses with proximity to the site and penalties for drifting away. As a secondary effect, this position-related reward helps reduce terminal vertical velocity. Since proximity to the target yields large bonuses, the agent is motivated to remain in these favorable positions, rather than at a higher altitude or larger horizontal displacement, to maximize the cumulative reward. To prevent indefinite hovering at low altitudes to exploit this bonus, a penalty for increasing altitude is added, along with a maximum time horizon and a bonus for successful landing. These measures help prevent the agent from "hacking" the reward function by indefinitely hovering to accumulate bonuses. Slowly approaching the landing site and landing gives a bonus larger than hovering above the target until the time horizon expires.

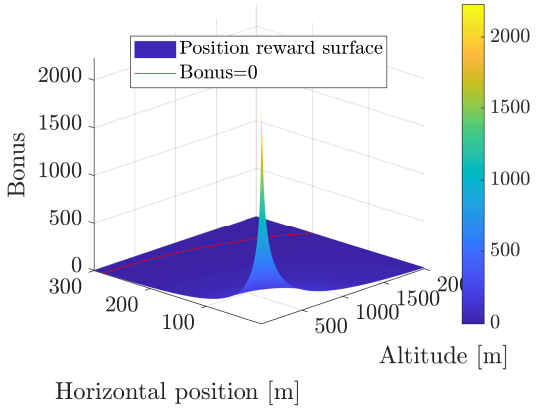


Fig. 3 Position reward function formulation.

Thanks to the scaling of each position component by its initial value, only position states very close to the landing site yield high rewards. As shown in Fig. 3, there are peak values near zero altitude and horizontal position, with a reward close to zero elsewhere, preventing the agent from "hacking" rewards by hovering and drifting away. For instance, given an initial position of (2000, 200, 200) m, landing 200 m from the target yields a normalized vector of (0, 1, 1). Hence, the argument of the logarithm is close to 1, giving an almost null reward, and encouraging the rocket to get closer to the landing site. As detailed in [30], the normalization with respect to the initial position prevents the agent from exploiting low-altitude states far from the target.

Training with a wrong position reward led the rocket to hover at low altitudes and drift away from the landing site. Adjusting the formulation with normalization corrected this issue, remarkably increasing the landing position precision.

B. Velocity

The velocity component of the reward function includes a penalty based on the deviation from a reference exponential velocity profile $\mathbf{v}_{ref}$ at each time step. This target profile guides the velocity components from their initial values to zero, and it is based on the rocket's position and the approximated time to go, which is estimated using the ratio between position and velocity vectors. The penalty is the norm of the error between the rocket and the reference velocities.

$$t_{go} = \frac{\|\mathbf{r}\|}{\|\mathbf{v}\|} \tag{36}$$

$$\mathbf{v}_{ref} = -\|\mathbf{v}_0\| \frac{\mathbf{r}}{\|\mathbf{r}\|} \left(1 - e^{-\frac{t_{go}}{\tau}}\right); \qquad \tau = 0.43 \tag{37}$$

$$R_{vel} = -0.01 \left\|\mathbf{v}_{ref} - \mathbf{v}\right\| \tag{38}$$

This approach is largely used in similar RL problems [31], where it has proven to be effective, but also led to suboptimal fuel consumption [7]. Given its suboptimal nature, its relative weight with respect to other terms in the total reward

function is kept lower. Since it only produces negative values, its significance diminishes as training progresses. Over time, other reward terms, such as the position component, provide substantially larger positive bonuses, making the velocity reward relatively less influential. Consequently, it only partially guides the velocity to follow the target reference profile and it is likely the least impactful term in the total reward function. Additionally, as mentioned earlier, the position reward also aids deceleration toward low-velocity states near the end of the flight.

C. Angular rates

The maximum angular rate for roll, pitch, and yaw is limited to 15 deg/s throughout the trajectory to prevent excessive rates and facilitate the decoupling of translational and rotational motion. Furthermore, high angular rates make it difficult for the fins and the TVC to control the vehicle, as actuator directions change rapidly. Additionally, excessive rates could induce fuel sloshing, not modeled in this work.

At each time step, a penalty is applied if the angular rates exceed this limit, otherwise, a bonus is given. Both are proportional to the difference from the constraint $\tilde{\omega}_{cstr}$.

$$R_{\|\omega\|} = -0.0006 \left[(\|\omega_x\| - \tilde{\omega}_{cstr}) + (\|\omega_y\| - \tilde{\omega}_{cstr}) + (\|\omega_z\| - \tilde{\omega}_{cstr}) \right] \tag{39}$$

This formulation discourages the agent from exceeding the maximum rates, stimulating it to minimize angular rates to avoid unwanted oscillations. However, the requirement is not strictly enforced, meaning that the total reward can still be large if angular rates are kept low for a major portion of the flight, even if the constraint of 15 deg/s is slightly exceeded at some time steps.

D. Vertical angle

Maintaining the rocket's correct orientation throughout the flight is crucial for stability, to prevent catastrophic failures. The goal is to keep a stable, near-vertical attitude so that the thrust counters gravity and downward velocity. In the **UEN** reference frame used, pitch and yaw are defined as the rotation around the horizontal y- and z-axes. An auxiliary angle "vertical angle" (δ) is defined as the angle between the $X_{\mathcal{B}}$-axis of the rocket's body frame and the vertical X_{UEN}-axis of the landing site:

$$\delta = \arccos\left(\cos\theta \cos\psi\right) \tag{40}$$

The objective is to prevent unstable orientations and encourage a pseudo-vertical attitude, balancing stability with maneuverability. The reward term is a piecewise function provided at each time step. A penalty is assigned when the vertical angle exceeds $\pi/2$ rad (i.e., 90 deg), proportional to how much the rocket points downward, as such orientations require significant correction. No reward is given for angles between 90 and 50 deg (i.e., $5\pi/18$ rad), while a fixed bonus is awarded below 50 deg. A constant bonus, instead of a proportional one, prevents encouraging a fully vertical orientation, which could reduce flexibility in correcting for horizontal displacement or tilted starting orientations.

$$R_\delta = \begin{cases} -0.02\left(\delta - \frac{\pi}{2}\right) & \text{if } \delta > \frac{\pi}{2} \\ 0 & \text{if } \frac{\pi}{2} > \delta > \frac{5\pi}{18} \\ +0.005, & \text{if } \delta < \frac{5\pi}{18} \end{cases} \tag{41}$$

E. Mass

The minimization of fuel consumption is the most straightforward term, where a penalty is given proportional to the mass difference at each time step.

$$R_{mass} = 2\left(m(t_k) - m(t_{k-1})\right) \tag{42}$$

F. Terminal reward

While the five described components are provided at each time step, the terminal reward is given only when the termination conditions are met and the flight ends. Since the fulfillment of the terminal constraints defines the landing success, the goal is to give hints to the agent as to whether it landed correctly or not. The terminal states are not only a consequence of the controls at the last time step but also of the sequence of previous actions performed during the flight.

For each terminal state, if the predefined threshold (outlined in the previous bullet points) is exceeded, the agent gets a quadratic negative penalty, proportional to how much it is violated. Conversely, if it fulfills the requirement, the agent

receives a logarithmic positive bonus, to suggest that achieving a terminal state of zero is the ideal outcome.

$$R_x = \begin{cases} k_1(x - \epsilon\tilde{x})^2 & \text{if } x > \epsilon\tilde{x} \\ k_2 \log\left(\frac{x}{\epsilon\tilde{x}}\right) & \text{if } x < \epsilon\tilde{x} \end{cases} \tag{43}$$

Logarithmic functions are used because they provide a theoretically infinite bonus for achieving an exact zero, unlike quadratic functions, which are capped. This strongly incentivizes the agent to prioritize states close to the desired values.

In Eq. (43), x describes the normalized terminal variable considered, while $\tilde{x}$ is the corresponding normalized terminal requirement. ϵ is a multiplicative factor that progressively enforces the terminal values, decreasing exponentially through the training. Introduced by [32, 33], this ϵ-constrained approach allows looser terminal constraints early in training when the agent has a more exploratory behavior and it has no clue on how to accomplish its tasks. In this phase, it helps the agent to understand the advantage of landing near target states. As training progresses, ϵ gradually tightens, enforcing stricter terminal conditions, to refine the agent's behavior to meet the exact thresholds. This approach has proven successful in similar space applications [15].

$$\epsilon_i = \begin{cases} \epsilon_0 & \text{if } i < i_0 \\ \epsilon_0 \left(\frac{\epsilon_f}{\epsilon_i}\right)^{\frac{i-i_0}{i_f}} & \text{if } i_0 < i < i_f \\ \epsilon_f & \text{if } i > i_f \end{cases} \tag{44}$$

ϵ_0 and i_0 represent the initial value and iteration when ϵ starts to decrease, while ϵ_f and i_f denote the final value and the iteration when the decrease stops. During the early training phase, where the agent explores the design space, larger constraints ($\epsilon_0 = 3$) are allowed. ϵ begins to reduce at iteration $i_0 = 1500$ once the agent's policy starts to be somehow defined. After around 70% of the training ($i_f = 7000$), ϵ becomes constant ($\epsilon_f = 1$) to fine-tune the policy.

For position, horizontal velocity, vertical velocity, vertical angle, and norm of angular rates there is a terminal reward term. Furthermore, there is a component for the landing altitude. It gives a fixed bonus if the rocket lands on the ground, while a penalty, proportional to the final altitude, is applied if the simulation terminates mid-air. Finally, an extra bonus is given if all the six constraints are fulfilled. Therefore, the final expressions for the terminal reward components are:

$$R_{r_{hor},terminal} = \begin{cases} -32\left(\left\|r_y, r_z\right\| - \epsilon\tilde{r}_{hor}\right)^2 & \text{if } \left\|r_y, r_z\right\| > \epsilon\tilde{r}_{hor} \\ -1.6\log\left(\frac{\left\|r_y, r_z\right\|}{\epsilon\tilde{r}_{hor}}\right) & \text{if } \left\|r_y, r_z\right\| < \epsilon\tilde{r}_{hor} \end{cases} \tag{45}$$

$$R_{v_{vert},terminal} = \begin{cases} -24\left(\|v_x\| - \epsilon\tilde{v}_{vert}\right)^2 & \text{if } \|v_x\| > \epsilon\tilde{v}_{vert} \\ -0.24\log\left(\frac{\|v_x\|}{\epsilon\tilde{v}_{vert}}\right) & \text{if } \|v_x\| < \epsilon\tilde{v}_{vert} \end{cases} \tag{46}$$

$$R_{v_{hor},terminal} = \begin{cases} -16\left(\left\|v_y, v_z\right\| - \epsilon\tilde{v}_{hor}\right)^2 & \text{if } \left\|v_y, v_z\right\| > \epsilon\tilde{v}_{hor} \\ -0.16\log\left(\frac{\left\|v_y, v_z\right\|}{\epsilon\tilde{v}_{hor}}\right) & \text{if } \left\|v_y, v_z\right\| < \epsilon\tilde{v}_{hor} \end{cases} \tag{47}$$

$$R_{\delta,terminal} = \begin{cases} -0.8\left(\delta - \epsilon\tilde{\delta}\right)^2 & \text{if } \delta > \epsilon\tilde{\delta} \\ -0.12\log\left(\frac{\delta}{\epsilon\tilde{\delta}}\right) & \text{if } \delta < \epsilon\tilde{\delta} \end{cases} \tag{48}$$

$$R_{\|\omega\|,terminal} = \begin{cases} -4\cdot 10^{-5}\left(\left\|\omega_x, \omega_y, \omega_z\right\| - \epsilon\|\tilde{\omega}\|\right)^2 & \text{if } \left\|\omega_x, \omega_y, \omega_z\right\| > \epsilon\|\tilde{\omega}\| \\ -0.04\log\left(\frac{\left\|\omega_x, \omega_y, \omega_z\right\|}{\epsilon\|\tilde{\omega}\|}\right) & \text{if } \left\|\omega_x, \omega_y, \omega_z\right\| < \epsilon\|\tilde{\omega}\| \end{cases} \tag{49}$$

$$R_{landing,terminal} = \begin{cases} -8r_x & \text{if } r_x > \tilde{r}_x \\ 0.8 & \text{if } r_x < \tilde{r}_x \end{cases} \tag{50}$$

$$R_{bonus,terminal} = \begin{cases} 10 & \text{if all constraints respected} \\ 0 & \text{otherwise} \end{cases} \tag{51}$$

The final expression for the terminal components is the sum of all the previously shown terms:

$$\begin{aligned} R_{terminal} = & R_{r_{hor},terminal} + R_{v_{vert},terminal} + R_{v_{hor},terminal} + R_{\delta,terminal} + R_{\|\omega\|,terminal} \\ & + R_{landing,terminal} + R_{bonus,terminal} \end{aligned} \tag{52}$$

Table 5 Meta-RL hyperparameters setup.

Parameter	Value
Batch size	8192
SGD minibatch size	128
SGD iterations	15
PPO clip parameter	0.01
Value function network	[256,128,128]
Number workers	8
Environment per worker	8
Activation function	tanh
Attention dimension	128
Number of heads	4
Memory length	250
Sequence length	128
Previous actions and rewards	250
Encoder MLP layers	[256,128,128]

G. Summary

A trial-and-error approach is used to tune the weights of each term in the reward function, similar to single-objective optimization where conflicting objectives are combined. For instance, minimizing fuel conflicts with reducing vertical velocity, as fuel is converted into thrust for deceleration. Thus, a balance between mass consumption and velocity must be found. Similarly, this applies to all the terms in the reward function.

During training, various metrics were monitored using `TensorBoard` to track trends and infer which features the agent was learning. If performance was unsatisfactory, the reward function weights were iteratively adjusted based on what was observed. This provided insights into whether the agent was minimizing some state variable errors at the expense of others. The final expression of the total reward function is the sum of the terms given at every time step and the ones given only at the end of the episode:

$$R = R_{pos} + R_{vel} + R_{\|\omega\|} + R_{\delta} + R_{mass} + \frac{\epsilon_0}{\epsilon} R_{terminal} \tag{53}$$

The term $\frac{\epsilon_0}{\epsilon}$ gradually increases the importance of terminal constraints relative to the dense reward. The dense reward only guides the agent early on toward the correct direction to have a meaningful policy, while terminal constraints are the real objectives to be optimized later in training.

Table 5 presents all the relevant parameters used in the neural network and in the RL algorithm, to reproduce the results. Furthermore, the learning rate is linearly interpolated between 10^{-4} and 10^{-6} between 0 and $8{\cdot}10^7$ time steps. The entropy coefficient is linearly interpolated between 10^{-4} and 0 between 0 and $4{\cdot}10^7$ time steps. Preliminary analysis on optimal trajectories showed that simulations last no more than 35 seconds, given the initial conditions in Table 1. Hence, the maximum time horizon in RL training is set to 40 seconds, to prevent "hacking" the reward function.

VI. Results

A. Meta-RL results

After training the agent for 9500 iterations, the solution is postprocessed and analyzed. As typically done in RL problems, once the training is finished and the network is deployed for analysis, the stochastic component is removed, making the policy deterministic [15]. Therefore, instead of outputting the probability of taking an action from a distribution, the network gives the expected value of the control action distribution. In PPO, since the action distributions are considered Gaussian, the output is the mean. Figure 4 presents the trend of the cumulative reward against the training iterations, as the training session progresses. Given the initial rapid rise and the plateauing after several iterations, it can

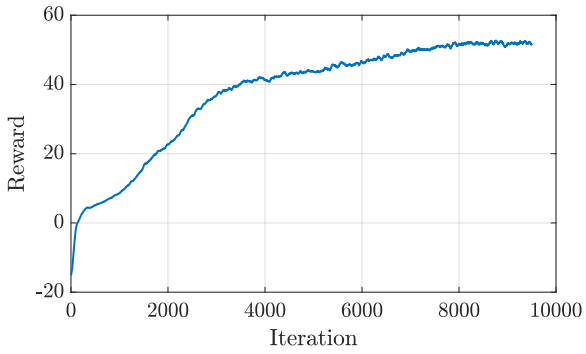


Fig. 4 Cumulative reward (moving average for 50 iterations) with respect to training iterations.

be determined that the NN has converged to a set of internal parameters. 1000 Monte Carlo simulations, considering all the uncertainties and initial dispersions, are run to evaluate the performance of the learned policy.

What can be seen in Table 6 is that the mean values of the terminal constraints of horizontal position, horizontal velocity, angular rates, and vertical angle are very close to zero, which is the desired goal. Also, the standard deviations are small, meaning that the terminal variables are not excessively dispersed, without significant outliers.

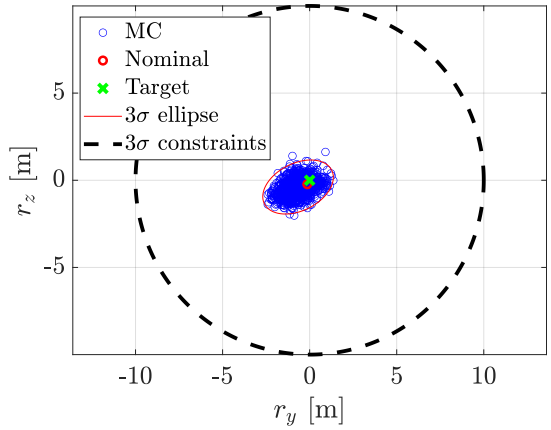
This explanation is supported by Fig. 5, where all the simulations successfully terminate close to the targets, inside the terminal constraints identified by the dashed lines, except for the vertical velocity. In Fig. 5a, it is observed that the terminal position is within the 10 m radius constraint, by a considerable margin, in all the 1000 runs. Similarly, Fig. 5b highlights that the terminal angular rates are well within the 3 deg/s boundaries. Figure 5c confirms the compliance of the guidance policy with respect to the terminal horizontal velocity of 1.5 m/s. All the simulations are within the requirement, with about 1 m/s margin. Likewise, Fig. 5d illustrates that the rocket always lands with a vertical angle significantly lower than the 3 deg constraints. The only terminal variable that exceeds its limit is the vertical velocity. While the rocket should land with a vertical velocity between -2 and 0 m/s, the mean value is about -3 m/s (Fig. 5e). Out of 1000 simulations, about 70 are compliant with the constraint, while the others have a maximum violation up to 3 m/s.

The slight violation is primarily due to conflicting terms in the reward function, requiring careful balancing to prevent certain variables from dominating others. For instance, by over-encouraging the reduction of the vertical velocity, the agent tends to exploit the reward function, persistently remaining in low altitude, low velocity states close to the landing site to maximize the cumulative reward over time. This behavior often exposes the rocket to wind influence, resulting in large TVC deflections to correct horizontal state errors. These corrections, however, significantly worsen other variables, especially those related to rotational motion: attitude and angular rates. Therefore, given the current reward function formulation, the neural network finds optimal to allow a slight violation of about 1 m/s of vertical velocity, to ensure compliance with constraints on other terminal variables.

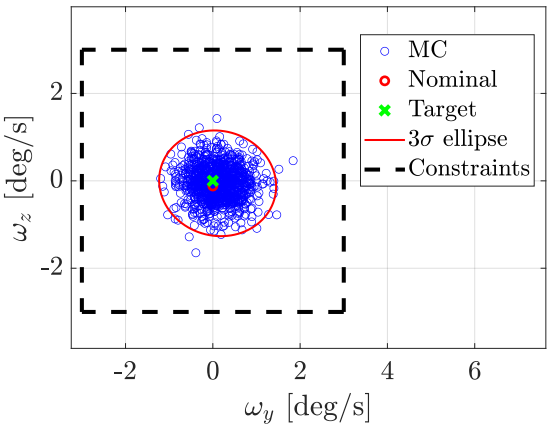
Unlike other states, influenced by thrust, TVC, and fins, the vertical velocity constraint is challenging because it is the only variable that solely depends on thrust magnitude, offering fewer control options. Moreover, the vertical velocity is "one-sided," exceeding the target value of 0 m/s only negatively. If it becomes positive, the rocket skips upward, making landing impossible unless the engine shuts down. This limitation leads the policy to conservatively prioritize other variables in such uncertain environments, avoiding the risk of skipping or worsening rotational states.

A simple, but effective, solution to this is to increase the thrust magnitude in the last few meters above the landing site (e.g., 2/3 m), creating a guidance mode for the vertical descent based on a solution to the uniformly accelerated motion. This correction substitutes the output of the neural network and it correctly solves the vertical velocity issue, eventually yielding 100% success rate, if the other terminal variables were already respected by a margin.

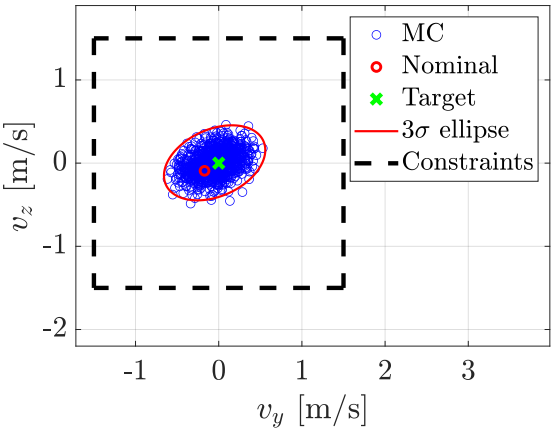
Figure 5f illustrates the trajectory flown for 1000 Monte-Carlo runs, highlighting large initial position dispersion. Despite this, a common path emerges across all flights. Initially, the rocket follows an inclined trajectory toward the horizontal position of the landing site. Once above it, a vertical descent is performed, controlling the rotational motion, while progressively reducing the vertical velocity and nullifying the horizontal one. Finally, right over the landing site, a final maneuver is performed to shrink the horizontal dispersion and land as close as possible to the target position. This clearly visible trend is mainly driven by the position component of the reward function, which encourages to take advantage of the high bonuses associated with states with small horizontal displacement. Even if not observable in this figure, the other terms of the reward function simultaneously incentivize to control rotational motion and reduce velocity.



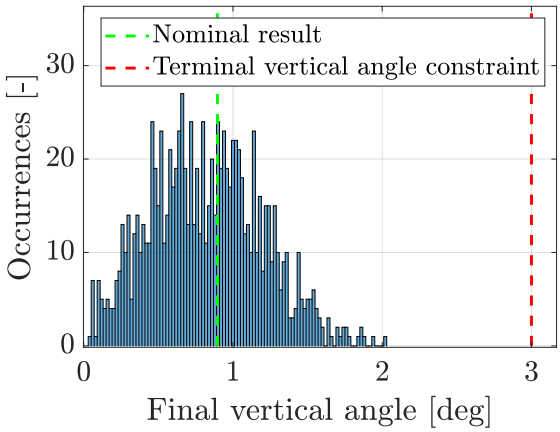
(a) Terminal horizontal position.



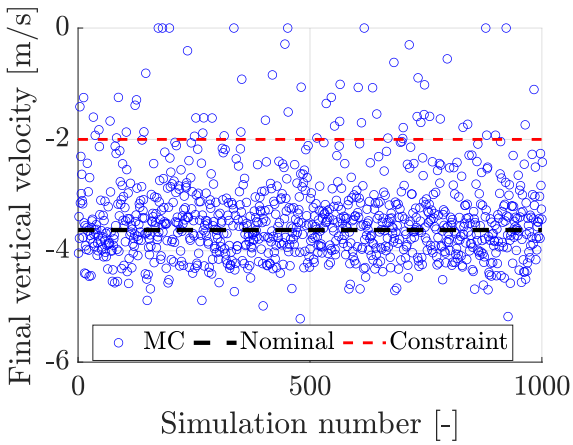
(b) Terminal angular rates.



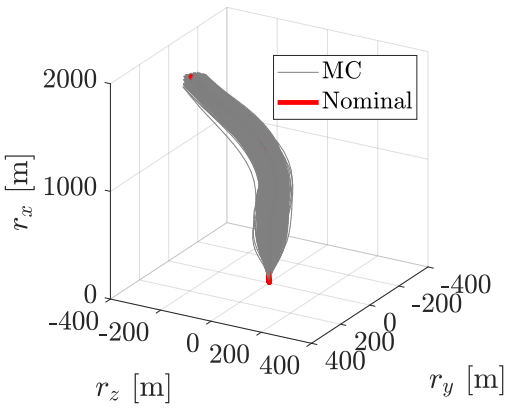
(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.



(f) 3D trajectories.

Fig. 5 Meta-RL GTrXL-based policy of 1000 Monte-Carlo simulations.

B. Comparison with LSTM

As mentioned in Sec. I, RNNs are typically used in many similar RL space guidance problems, for their easy implementation. In this work, it has been hypothesized that Transformer-based NNs would outperform LSTMs in complex, high-dimensional G&C problems, such as the one considered here. To confirm this hypothesis, a new training session is done, substituting GTrXL with LSTM, to perform a comparison between the two policies.

LSTM is a type of recurrent neural network that uses feedback connections and hidden states to retain information about previous states and actions, acting as a sort of memory. Therefore, each new action is executed based on the current observation and on an abstract representation of previous ones. This network's architecture is very similar to the GTrXL one: there are three encoder layers, and the LSTM module that substitutes GTrXL has a cell size of 256.

The results presented in Table 6 reveal that the terminal dispersions with LSTM are significantly larger than with GTrXL, especially affecting angular rates and vertical angle, with mean values two to seven times worse than the Transformer results. Even more eye-catching are the large dispersions of these terminal states, giving a huge number of failed simulations, with substantial terminal errors. Despite having a similar mean value, the terminal horizontal and vertical velocity dispersions are about three larger, yielding many outliers outside of the terminal constraints. Only the terminal position given by the LSTM policy is in line with the GTrXL one.

This clearly shows that LSTM-based NNs struggle to effectively capture the different aspects and relationships of such a complex problem. It is an evident limit of these kinds of networks, where previous information is compressed in only two internal hidden states, which are not enough to build a secondary policy to handle all the uncertainties and dispersions. Moreover, the training time almost doubles with respect to the Transformer-based policy. Not considering the always exceeded vertical velocity constraint, only 489 out of 1000 simulations fulfill all the other terminal constraints. For the GTrXL policy, the compliance rate was 100%.

Thus, the low success rate, the larger mean, and the dispersion of terminal variables prove that LSTM networks are an unsuitable choice for such a problem, largely underperforming compared to Transformer-based neural networks.

C. Comparison with conventional guidance and control strategy

The RL-based solution is compared to a conventional guidance and control strategy, made of a Linear Quadratic Regulator (LQR) controller that tracks an optimal trajectory. The optimal trajectory is generated using nominal dynamics, initial conditions, and without uncertainties or winds. ICLOCS2 ([34]) is used to generate it, using a Hermite-Simpson direct collocation method to transcribe the infinite-dimensional Optimal Control Problem (OCP) into a large, sparse Non-Linear Programming (NLP) problem. The goal is to minimize fuel consumption, performing a pinpoint landing while satisfying the EOMs, and a set of state and control constraints at the discrete collocation points [35, 36].

An LQR controller is designed to track the nominal trajectory when perturbations are introduced. It is based on optimal control theory for linear systems, where a linear quadratic cost function is minimized, balancing state errors and control efforts. $\mathbf{Q}$ and $\mathbf{R}$ are weighting matrices to balance how much state errors and control effort are allowed, affecting the controller response. The gains obtained are multiplied by the state deviations, to compute the control actions, in a state-feedback fashion. The Bryson's rule method [37] is used to tune $\mathbf{Q}$ and $\mathbf{R}$ for the required performance.

The controller architecture consists of two loops. The outer loop adjusts the translational motion providing the trajectory corrective thrust magnitude and desired attitude, while the inner loop controls the rotational motion stabilizing the attitude and angular rates using TVC and fins deflections. Since the LQR does not enforce any type of constraint by definition, a numerical rate limiter and a deflection saturation are applied to the control variables.

The process used to design the controller is inspired by the one used in [21], iteratively tuning the $\mathbf{Q}$ and $\mathbf{R}$ matrices. First, the controller's performance is assessed with linear analysis, such as step response on the nominal trajectory for the two loops independently. Then, 6-DOF Monte-Carlo simulations are performed, introducing dispersed initial conditions, winds, and uncertainties in the dynamics. The robustness of the G&C policy is tested against external perturbations, which are not embedded neither in the nominal trajectory (guidance), nor in the LQR design (control).

This conventional strategy shows high sensitivity to gain changes due to the coupling between translational and rotational motion. For instance, the inner loop controls rotational motion by translating attitude and angular rate errors into TVC deflection angles to induce thrust moments, but their effectiveness also depends on thrust magnitude, which is commanded by the outer loop. Therefore, while gains are designed independently, a clear interdependence exists between the loops. If the thrust magnitude is increased, becoming larger than the reference one used to calculate the TVC deflections, it can create excessive moments that the fins alone cannot counteract due to their limited surface area and low dynamic pressure. This can cause instability, with the vehicle potentially flipping or spinning uncontrollably.

To address this, less aggressive controllers can be designed, allowing larger state errors and limiting control efforts. This reduces thrust increases in the outer loop and constraints TVC deflections, minimizing unexpected moments but propagating larger errors, leading to more frequent terminal constraint violations. Tuning the weight matrices to balance terminal accuracy and flight stability is a challenging and iterative process.

After tuning the controller extensively, acceptable Monte-Carlo results were obtained, as a trade-off between the two aforementioned principles. Out of 1000 Monte-Carlo runs, they all manage to track the nominal trajectory to a

Table 6 Mean and standard deviation of terminal variables from different policies.

Training	Norm Horizontal Position [m]		Norm Horizontal Velocity [m/s]		Vertical Velocity [m/s]		Norm Angular rates [deg/s]		Vertical angle [deg]		Mass Consumption [kg]	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
GTrXL	1.01	0.55	0.23	0.12	-3.39	0.80	0.55	0.28	0.82	0.37	536.4	21.7
LSTM	0.79	0.76	0.55	0.42	-3.92	2.31	4.39	3.49	2.11	2.06	506.4	24.1
OCP+LQR	6.94	2.18	1.45	0.40	-0.53	0.61	3.73	1.73	0.55	0.25	502.3	11.5

certain extent, with no one that diverges significantly. This is remarkable since LQR is not a robust control technique because it does not account for model uncertainties and disturbances. However, terminal constraint performance is worse than the meta-RL policy, as shown in Table 6. While terminal attitude and vertical velocity remain within limits, position, horizontal velocity, and angular rates often exceed the constraints, primarily due to winds and dispersed initial conditions. For example, the wind profiles introduce a bias on the horizontal position and velocity, leading the G&C strategy to generate an excessive thrust to compensate, making the rocket unstable in some cases. As a trade-off, to avoid failed flights, larger errors are tolerated, resulting in only 390 out of 1000 simulations meeting all terminal constraints.

The conventional approach performs poorly due to the interactions between translational and rotational motions, separately controlled by two 3-DOF loops. This separation proved to be suboptimal for tightly coupled interactions. It performs better only in terms of fuel consumption, relying on a minimum mass optimal trajectory. By contrast, RL inherently accounts for all the interactions during training, enabling the NN to learn how actions impact translational and rotation motions to generate policies that consider their combined effects. Although the RL-based approach shows about 7% larger fuel consumption, it outperforms the conventional G&C strategy in meeting the terminal constraint.

Moreover, LQR uses a linearized model around states and controls from a nominal trajectory, without considering uncertainties during the synthesis process. Conversely, the RL problem is trained on the non-linear dynamics, including uncertainties and initial dispersions, so that the NN experiences all possible effects and learns a robust policy.

VII. Conclusions

In this research, Meta-Reinforcement Learning is combined with Gated Transformer XL attention-based neural networks to obtain a robust integrated guidance and control policy for atmospheric rocket landing. Thanks to the GTrXL capabilities to leverage past experiences and to find relationships between elements in long sequences, the G&C policy learned by the neural network is able to successfully land the rocket in such a challenging and uncertain environment.

Designing a reward function for complex problems with many states and controls is challenging, as it requires balancing conflicting objectives. The successful approach used in this work comprises dense rewards (bonuses/penalties at each time step), to drive the vehicle towards the target states, and episodic rewards (given at the episode's end), to encourage compliance with terminal constraints. Logarithmic expressions for dense rewards are effective when a state has an attraction point, such as guiding the rocket's position to the landing site. For episodic rewards, quadratic penalties for violations and logarithmic bonuses for compliance are effective, pushing the NN to stay within limits with a margin.

The meta-RL policy performs well in Monte-Carlo campaigns, fulfilling terminal constraints on position, horizontal velocity, angular rate, and vertical angle in 1000 out of 1000 runs. The vertical velocity is the only constraint slightly violated: while it should be between 0 and -2 m/s, the mean value is slightly below -3 m/s, with about 70 simulations respecting the limit. The NN-based policy also shows good potential to run in real-time, because a forward pass to compute the control actions takes about 6 ms on a computer equipped with an Intel® Core™ i7-8565U processor.

Furthermore, the GTrXL-based policy significantly outperforms the LSTM-based one in terms of terminal constraint variables. Although both exhibit a slight violation of the vertical velocity constraint, the LSTM-based policy shows a greater number of outliers (confirmed by a larger standard deviation), achieving only a 48.9% success rate on the other terminal variables, compared to 100% success rate of the GTrXL policy. This degradation is traced back to the architectural differences between the two neural networks. While attention-based networks, such as GTrXL, process sequences in parallel, with many more internal parameters to retain past information, RNNs, such as LSTM, process elements sequentially, and compress a representation of the past information in only two internal states of fixed dimension. Therefore, in such complex, uncertain, and high-dimensional problems, these differences prevent LSTM from effectively creating a policy that captures time relationships, to execute optimal actions based on a history of events.

Additionally, the Transformer-based policy clearly outperforms the considered G&C conventional strategy made

of an optimal trajectory tracked by an LQR controller. First of all, unlike the meta-RL approach, the conventional strategy is not a robust method, not embedding any kind of uncertainty into the design process. Secondly, the LQR independently controls the translational and rotational motion, while the meta-RL method concurrently controls the 6-DOF, effectively understanding and managing interactions between state variables, thanks to the training experience.

Future works include the optimization of the reward function to effectively address the slight terminal vertical velocity violation, for example encouraging the network to learn a piece-wise policy, with a first portion identical to the one learned in this work, and a second segment forcing a vertical motion in the last few meters above the landing site. The addition of more accurate local wind models at low altitudes is foreseen as a future improvement, together with the inclusion of sloshing and flexible vehicle models to further enhance the accuracy and the fidelity of the environment adopted to shape the feedback policy to be learned.

Acknowledgments

The first author would like to thank the Department of Guidance, Navigation, and Control at the German Aerospace Center (DLR) in Bremen, Germany, for hosting him as an intern to carry out his Master's Thesis research.

The authors acknowledge the use of computational resources of the DelftBlue supercomputer, provided by Delft High Performance Computing Centre (`https://www.tudelft.nl/dhpc`).

References

[1] Acikmese, B., and Ploen, S. R., "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 5, 2007, pp. 1353–1366. https://doi.org/10.2514/1.27553.

[2] Lu, P., "Propellant-Optimal Powered Descent Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 4, 2018, pp. 813–826. https://doi.org/10.2514/1.G003243.

[3] Reynolds, T., Malyuta, D., Mesbahi, M., Acikmese, B., and Carson, J. M., "A Real-Time Algorithm for Non-Convex Powered Descent Guidance," *AIAA Scitech 2020 Forum*, 2020. https://doi.org/10.2514/6.2020-0844.

[4] Sagliano, M., Seelbinder, D., Theil, S., and Lu, P., "Six-Degree-of-Freedom Rocket Landing Optimization via Augmented Convex–Concave Decomposition," *Journal of Guidance, Control, and Dynamics*, Vol. 47, No. 1, 2024, pp. 20–35. https://doi.org/10.2514/1.G007570.

[5] Sagliano, M., Heidecker, A., Hernández, J. M., Farì, S., Schlotterer, M., Woicke, S., Seelbinder, D., and Dumont, E., *Onboard Guidance for Reusable Rockets: Aerodynamic Descent and Powered Landing*, 2021. https://doi.org/10.2514/6.2021-0862.

[6] Sagliano, M., Heidecker, A., Farì, S., Alfredo, M. H. J., Schlotterer, M., Woicke, S., Seelbinder, D., and Dumont, E., *Powered Atmospheric Landing Guidance for Reusable Rockets: the CALLISTO studies*, 2024. https://doi.org/10.2514/6.2024-1761.

[7] Gaudet, B., Linares, R., and Furfaro, R., "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Landing," *Advances in Space Research*, Vol. 65, No. 7, 2020, pp. 1723–1741. https://doi.org/10.1016/j.asr.2019.12.030.

[8] Federici, L., Scorsoglio, A., Zavoli, A., and Furfaro, R., "Autonomous Guidance Between Quasiperiodic Orbits in Cislunar Space via Deep Reinforcement Learning," *Journal of Spacecraft and Rockets*, 2023, pp. 1–12. https://doi.org/10.2514/1.A35747.

[9] Federici, L., Benedikter, B., and Zavoli, A., "Deep Learning Techniques for Autonomous Spacecraft Guidance During Proximity Operations," *Journal of Spacecraft and Rockets*, Vol. 58, No. 6, 2021, pp. 1774–1785. https://doi.org/10.2514/1.A35076.

[10] Federici, L., Scorsoglio, A., Zavoli, A., and Furfaro, R., "Meta-Reinforcement Learning for Adaptive Spacecraft Guidance during Finite-Thrust Rendezvous Missions," *Acta Astronautica*, Vol. 201, 2022, pp. 129–141. https://doi.org/10.1016/j.actaastro.2022.08.047.

[11] Gaudet, B., Linares, R., and Furfaro, R., "Six Degree-of-Freedom Body-Fixed Hovering over Unmapped Asteroids via LIDAR Altimetry and Reinforcement Meta-Learning," *Acta Astronautica*, Vol. 172, 2020, pp. 90–99. https://doi.org/10.1016/j.actaastro.2020.03.026.

[12] Federici, L., Scorsoglio, A., Ghilardi, L., D'Ambrosio, A., Benedikter, B., Zavoli, A., and Furfaro, R., "Image-Based Meta-Reinforcement Learning for Autonomous Guidance of an Asteroid Impactor," *Journal of Guidance, Control, and Dynamics*, Vol. 45, No. 11, 2022, pp. 2013–2028. https://doi.org/10.2514/1.G006832.

[13] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I., "Attention Is All You Need," , Aug. 2023. ArXiv:https://doi.org/10.48550/arXiv.1706.03762.

[14] Parisotto, E., Song, H. F., Rae, J. W., Pascanu, R., Gulcehre, C., Jayakumar, S. M., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M. M., Heess, N., and Hadsell, R., "Stabilizing Transformers for Reinforcement Learning," , Oct. 2019. ArXiv:https://doi.org/10.48550/arXiv.1910.06764.

[15] Federici, L., and Furfaro, R., "Meta-Reinforcement Learning with Transformer Networks for Space Guidance Applications," *AIAA SCITECH 2024 Forum*, American Institute of Aeronautics and Astronautics, 2024. https://doi.org/10.2514/6.2024-2061.

[16] Sutton, R. S., and Barto, A., *Reinforcement Learning: An Introduction*, Adaptive Computation and Machine Learning, The MIT Press, Cambridge, Massachusetts, 2018.

[17] Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S., "A Survey of Meta-Reinforcement Learning," , Jan. 2023. https://doi.org/arXiv:2301.08028.

[18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms," , Aug. 2017. ArXiv: https://doi.org/10.48550/arXiv.1707.06347.

[19] Simplício, P., Marcos, A., and Bennani, S., "Reusable Launchers: Development of a Coupled Flight Mechanics, Guidance, and Control Benchmark," *Journal of Spacecraft and Rockets*, Vol. 57, No. 1, 2020, pp. 74–89. https://doi.org/10.2514/1.A34429.

[20] Simplício, P., Marcos, A., and Bennani, S., "Guidance of Reusable Launchers: Improving Descent and Landing Performance," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 10, 2019, pp. 2206–2219. https://doi.org/10.2514/1.G004155.

[21] Sagliano, M., Hernández, J. A. M., Farì, S., Heidecker, A., Schlotterer, M., Woicke, S., Seelbinder, D., Krummen, S., and Dumont, E., "Unified-Loop Structured H-Infinity Control for Aerodynamic Steering of Reusable Rockets," *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 5, 2023, pp. 815–837. https://doi.org/10.2514/1.G007077.

[22] Guédron, S., Ishimoto, S., Dumont, E., Tatiossian, P., Chavagnac, C., Desmariaux, J., Monchaux, D., Frenoy, O., Moreno, E. C., Deremaux, C., Lidon, N., Cesco, N., Witte, L., Sagliano, M., Seelbinder, D., Klevanski, J., Ecker, T., Reimann, B., Riehmer, J., Ertl, M., and Krummen, S., "CALLISTO DEMONSTRATOR: Focus on system aspects," *71th International Astronautical Congress, Dubai, UAE*, , No. IAC-20-D2.6.1, 2021.

[23] Lee, S.-D., and Lee, C.-H., "Multi-Phase and Dual Aero/Propulsive Rocket Landing Guidance using Successive Convex Programming," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 237, No. 8, 2022, pp. 1816–1834. https://doi.org/10.1177/09544100221138350.

[24] De Oliveira, A., and Lavagna, M., "Development of a Controlled Dynamics Simulator for Reusable Launcher Descent and Precise Landing," *Aerospace*, Vol. 10, No. 12, 2023, pp. 993–1022. https://doi.org/10.3390/aerospace10120993.

[25] Sagliano, M., Seelbinder, D., Theil, S., Im, S., Lee, J., and Lee, K., "Booster Dispersion Area Management through Aerodynamic Guidance and Control," *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics, San Diego, CA & Virtual, 2022. https://doi.org/10.2514/6.2022-0759.

[26] NOOA, USAF, and NASA, "US. Standard Atmosphere, 1976," Tech. rep., 1976.

[27] Drob, D. P., Emmert, J. T., Meriwether, J. W., Makela, J. J., Doornbos, E., Conde, M., Hernandez, G., Noto, J., Zawdie, K. A., McDonald, S. E., Huba, J. D., and Klenzing, J. H., "An update to the Horizontal Wind Model (HWM): The quiet time thermosphere," *Earth and Space Science*, Vol. 2, No. 7, 2015, pp. 301–319. https://doi.org/10.1002/2014EA000089.

[28] Leahy, F., "Discrete Gust Model for Launch Vehicle Assessments," *AMS Annual Meeting 12th Conference on Aviation, Range and Aerospace Meteorology*, 2008.

[29] Bonasera, S., Bosanac, N., Sullivan, C. J., Elliott, I., Ahmed, N., and McMahon, J. W., "Designing Sun–Earth L2 Halo Orbit Stationkeeping Maneuvers via Reinforcement Learning," *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 2, 2023, pp. 301–311. https://doi.org/10.2514/1.G006783.

[30] Carradori, J., "6-DOF Atmospheric Rocket Landing Guidance using Meta-Reinforcement Learning," Master's thesis, TU Delft, 2024. URL http://resolver.tudelft.nl/uuid:bf2a598c-9694-40cd-8dec-03b73d539b54.

[31] Gaudet, B., Linares, R., and Furfaro, R., "Adaptive Guidance and Integrated Navigation with Reinforcement Meta-Learning," *Acta Astronautica*, Vol. 169, 2020, pp. 180–190. https://doi.org/10.1016/j.actaastro.2020.01.007.

[32] Federici, L., and Zavoli, A., "Robust Interplanetary Trajectory Design under Multiple Uncertainties via Meta-Reinforcement Learning," *Acta Astronautica*, Vol. 214, 2024, pp. 147–158. https://doi.org/10.1016/j.actaastro.2023.10.018.

[33] Zavoli, A., and Federici, L., "Reinforcement Learning for Robust Trajectory Design of Interplanetary Missions," *Journal of Guidance, Control, and Dynamics*, Vol. 44, No. 8, 2021, pp. 1440–1453. https://doi.org/10.2514/1.G005794.

[34] Nie, Y., Faqir, O., and Kerrigan, E., "ICLOCS2: Solve your optimal control problems with less pain," , 2018.

[35] Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed., Society for Industrial and Applied Mathematics, 2010. https://doi.org/10.1137/1.9780898718577.

[36] Sagliano, M., Theil, S., Bergsma, M., D'Onofrio, V., Whittle, L., and Viavattene, G., "On the Radau pseudospectral method: theoretical and implementation advances," *CEAS Space Journal*, Vol. 9, No. 3, 2017, pp. 313–331. https://doi.org/10.1007/s12567-017-0165-5.

[37] Bryson, A., "Applied Optimal Control: Optimization, Estimation and Control," *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 59, No. 8, 1979, p. 402. https://doi.org/https://doi.org/10.1002/zamm.19790590826.