Image-based Deep Reinforcement Meta-Learning for Autonomous Lunar Landing

Andrea Scorsoglio*
Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721

Andrea D'Ambrosio†
School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT
Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721

Luca Ghilardi‡ and Brian Gaudet§
Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721

Fabio Curti ¶
School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT

Roberto Furfaro ‖
Department of System & Industrial Engineering, Department of Aerospace & Mechanical Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721

Future exploration and human missions on large planetary bodies (e.g. Moon, Mars) will require advanced guidance navigation and control algorithms for the powered descent phase, which must be capable of unprecedented levels of autonomy. The advent of machine learning, and specifically reinforcement learning, has enabled new possibilities for closed-loop autonomous guidance and navigation. In this paper we apply image-based reinforcement meta-learning to solve the lunar pinpoint powered descent and landing task with uncertain dynamical parameters and actuator failure. The agent, a deep neural network, takes real-time images and ranging observations acquired during the descent and maps them directly to thrust command (i.e. sensor-to-action policy). Training and validation of the algorithm and Monte Carlo simulations shows that the resulting closed-loop guidance policy reaches errors in the order of meters in different scenarios, even when the environment is partially observed, and the state of the spacecraft is not fully known.

*PhD Student, Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721
†Ph.D. Candidate, School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT
‡Ph.D. Student, Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721
§Research Engineer, Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721
¶Professor, School of Aerospace Engineering, Sapienza University of Rome, Via Salaria 851, 00138 Rome, IT
‖Professor, Department of System & Industrial Engineering, Department of Aerospace & Mechanical Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721
Previously presented at *AIAA Scitech 2020 Forum*, Orlando, FL, January 6-10, 2020, p. 1910

Nomenclature

$\mathcal{A}$	=	action space
α	=	velocity error reward coefficient
$A^{\pi}(\mathbf{x}_k, \mathbf{u}_k)$	=	advantage function
β_{θ}	=	policy learning rate
β_w	=	critic learning rate
δ	=	bound on policy update
η	=	positive reward constant
$\boldsymbol{g}$	=	gravity vector [m/s^2]
γ	=	discount factor
$\boldsymbol{I}$	=	image observation
I_{sp}	=	specific impulse [s]
$J(\theta)$	=	objective function
κ	=	position error coefficient
$L(\theta)$	=	proximal policy optimization objective function
$L(w)$	=	critic cost function
m	=	spacecraft mass [kg]
$\mathcal{M}$	=	markov decision process
$O(\boldsymbol{x})$	=	observation function
o_k	=	observation at step k
o_{VF}	=	observation - value function
$o_{\pi_{\theta}}$	=	observation - value function
π_{θ}	=	policy
$\mathrm{P}(x_{k+1}\|x_k, u_k)$	=	state transition distribution
$p_k(\theta)$	=	policy probability ratio
$Q^{\pi}(\mathbf{x}_k, \mathbf{u}_k)$	=	state-action value function
$\mathrm{R}(x_k, u_k)$	=	reward function
r_k	=	reward signal at step k
$\boldsymbol{r}$	=	position vector [m]
$\boldsymbol{r}_0$	=	initial position [m]
$\boldsymbol{r}_f$	=	final position [m]
$\boldsymbol{r}_t$	=	target position [m]

$\boldsymbol{r}_{t'}$	=	reference position above target for reward
$r_{lim,1}$	=	left limit position interval
$r_{lim,2}$	=	right limit position interval
r_x	=	position x component [m]
r_y	=	position y component [m]
r_z	=	position z component [m]
$\mathcal{S}$	=	state space
$\boldsymbol{T}$	=	thrust [N]
$\boldsymbol{\tau}$	=	trajectory
τ_1	=	reward time constant 1 [s]
τ_2	=	reward time constant 2 [s]
t_{go}	=	time-to-go
T_x	=	thrust x component [N]
T_y	=	thrust y component [N]
T_z	=	thrust z component [N]
T_{min}	=	minimum thrust [N]
T_{max}	=	maximum thrust [N]
$\boldsymbol{\theta}$	=	policy parameters
$\boldsymbol{u}_k$	=	control at step k
$V^{\pi}(\mathbf{x}_k)$	=	value function
$\boldsymbol{v}_{t'}$	=	reference velocity above target for reward
$v_{lim,1}$	=	left limit velocity interval
$v_{lim,2}$	=	right limit velocity interval
$\boldsymbol{v}_0$	=	initial velocity [m/s]
$\boldsymbol{v}_f$	=	final velocity [m/s]
$\boldsymbol{v}_{err}$	=	velocity error
$\boldsymbol{v}_t$	=	target velocity [m/s]
x_k	=	markov decision process state at step k
ξ	=	velocity error coefficient

I. Introduction

PLANETARY autonomous landing is becoming increasingly important for solar system exploration. With the Artemis program renewing the interest in the Moon[1] and the ongoing Mars exploration program[2], a new level of autonomy will be needed for the next generation of spacecraft and landers. Such autonomous systems must be able to integrate critical Guidance, Navigation and Control (GNC) functions, such as algorithms for precision or pinpoint landing (< 10 meters accuracy), with hazard avoidance approaches to enable safe and accurate landing on the desired location on the planetary surface. More specifically, the GNC system must be able to 1) process the data in real-time to navigate during the powered descent toward the surface and 2) execute maneuvers conducive to pinpoint-landing while enabling real-time decision making that ensures safe landing on the selected planetary location. In standard spacecraft architectures, the overall GNC functions are decoupled and designed separately. The navigation subsystem determines position and velocity of the lander from sensors information, whereas the guidance subsystem determines/computes the appropriate level of thrust and its direction as function of the current state (i.e. lander position and velocity). Finally the control subsystem is responsible for implementing the desired guidance command in a closed-loop fashion.

Guidance algorithms are generally composed of two major segments, i.e. a) a targeting algorithm and b) a trajectory-following, real-time guidance algorithm. The targeting algorithm computes the reference trajectory driving the lander to the lunar surface. The latter is generally computed to ensure minimum fuel while satisfying appropriate thrust and path constraints. Conversely, the real-time guidance algorithm computes the acceleration command that must be implemented by the lander thrusters to track the reference trajectory for a precise and soft landing. The original Apollo real-time targeting and guidance algorithm [3], was based on an iterative method that generated a nominal trajectory consisting in a quartic polynomial. Importantly, the feedback Apollo real-time guidance was derived by approximating the nominal trajectory by a 4th-order McLaurin expansion of the reference trajectory [3, 4]. The methodologies used today are in general based on variations of the Apollo-era off-line procedures. A reference trajectory is usually computed on the ground (man in the loop) and successively given as input to the lander's guidance and control algorithm. Normally, the off-line reference trajectory is computed by solving an Optimal Control Problem (OCP). This can be achieved through two different approaches: indirect and direct methods. In indirect methods, the calculus of variations is exploited together with the Pontryagin Minimum Principle to retrieve the optimal control and the first order necessary conditions from the Hamiltonian of the problem. This procedure leads to writing a Two Point Boundary Value Problem (TPBVP), which is made up of a set of differential equations in terms of states and costates. Commonly, TPBVPs are solved via shooting methods [5]. However, although this approach provides very accurate optimal solutions, it suffers from the high sensitivity of the initial guesses on the optimal solution. In order to overcome this limitation, novel and accurate approaches have been proposed, including Theory of Functional Connections (TFC) [6] and the Universal Powered Guidance (UPG) [7]. The first has been used to solve some space guidance problems, such as energy [8] and fuel [9] optimal landing trajectories on Moon and Mars. The crucial results of these papers have been the

machine-level accuracy of the solution and the exceptionally low computational time, which makes this approach very suitable for a future on-board implementation. The second, develops a generalized optimal powered guidance approach to obtain the optimal guided trajectory for a powered descent on large planetary bodies (e.g. precision landing systems to deliver large spacecraft with humans on the surface of Mars [7])

On the other hand, direct methods relies on the discretization of the original continuous OCP into a finite constrained optimization problem. This leads to the transcription of continuous problem into a Non-linear Programming (NLP) Problem [10], which is usually solved via traditional algorithms, such as the interior point [11], the trust region [12] or the Nelder-Mead [13] algorithms. These approaches are very computationally demanding and they do not provide any guarantees about the convergence to the optimal solution if the problem is non-convex. However, the so-called convexification technique [14] has been recently proposed to transform non-convex problems into convex problems, whose convergence to the optimal solution is guaranteed in a finite number of steps, and therefore they can be solved more easily and faster. Many works have employed such a method to obtain fast and accurate optimal powered descent trajectories in large and small planetary bodies [15–18].

As introduced above, the guidance algorithms, regardless of the architecture, must be integrated with the navigation system, which is responsible for determining lander position and velocity. The most common approach to Relative Terrain Navigation (RTN) [19] is to define algorithms that can process optical images to reconstruct the spacecraft state in real-time. The relative position and velocity are estimated on-board by extracting and correlating/registering landmarks on the planetary bodies [20] and tracking them across frames. An example of such technique is called Natural Feature Tracking (NFT) which was used on the ongoing mission OSIRIS-REx [21]. Recently, studies have demonstrated that a spacecraft can navigate by tracking unknown lunar surface features without ever solving for the 3-D location of those surface features [22].

The aforementioned methods summarize the state-of-the-art in trajectory optimization and navigation for powered descent and landing. However, they have a major drawback in that they are not robust against un-modeled dynamics as well as against uncertainties on lander state estimation. Consequently, the ability of the system to quickly respond and adapt, is limited. Moreover, both direct and indirect methods are designed to compute open-loop trajectories, which means that a navigation algorithm and a control modules are required to effectively track those trajectories. New methods have been proposed to fastly compute optimal trajectories on-board to overcome these limitations, potentially enabling new levels of autonomy and precision. This also complies with NASA's Vision for Space Exploration program [23], which imposes a new set of requirements to automatize the tasks requested for the planetary landing to minimize the crew members' routine tasks. Within this framework, adaptive guidance is exploited by tuning its parameters depending on the specific environment. One interesting example of this approach is the Chinese lander Chang'e 3 that successfully landed on the Moon using a combination of optical and radar sensors [24]. The lander in this case selects the landing site dynamically to meet the hazardous requirements in complete autonomy. The law adopted between

$\sim$ 2.4km and $\sim$ 100m is a modified polynomial guidance, which allows fast trajectory computation.

Over the past few years, there has been a growing interest in Machine Learning (ML) as applied to space GN&C. In the past decade, the interest in ML has increased, also due to its application in many fields, including computer vision, natural language processing, robotics, to mention a few. Many of the recent ML approaches come from the past century but the relatively low computational power then available has limited its fields of application. With the new millennium and a rapid increase in computational power availability, both with massive parallelization of CPUs and GPUs, ML and neural networks specifically, have seen a major developments. Today, ML is ubiquitous; it is used both for data analysis and for robotic motion tasks. In the last few years, ML has gained recognition as a method to solve adaptive spacecraft guidance problems, but it is still underutilized. There have been some early examples of deep neural networks being trained using supervised learning to compute landing trajectories on Mars [25], on the Moon [26, 27], on asteroids [28], and for interplanetary trajectories together with evolutionary techniques in [29]. Such approaches assume one has access to the complete knowledge of the spacecraft state. To overcome this limitation, deep convolutional neural networks have been used in [30] in a supervised fashion to map the thrust from a sequence of images for 1D and 2D lunar landing trajectories, effectively removing the need to know the spacecraft state. The dataset was created using fuel optimal trajectories generated via GPOPS [31].

Although these examples prove the viability of machine learning for trajectory optimization, in the recent years it has become apparent that classical neural networks trained via supervised learning are not sufficient to achieve high level of accuracy in uncertain environments with complex constraints. Simultaneously, reinforcement learning (RL) [32], the third branch of machine learning, has started to gain recognition. Differently from standard supervised learning for trajectory optimization, RL optimizes a parametrized policy by direct interaction with a simulated or real environment without a teacher (i.e. training set). Recently, the field of RL has experienced a dramatic growth with multiple breakthrough discoveries in the last decade. RL agents have been able to solve complex tasks [33], even when the input space is high dimensional (i.e. image input) [34]. In the last few years, a multitude of applications to spacecraft guidance have been developed, ranging from planetary landing [35–37], to path planning for asteroid hopping rovers [38], rendezvous [39–41], low thrust trajectory design [42–45] and formation flying [46]. RL is capable of creating guidance laws in complex environments in all these applications, even with non-convex constraints scenarios [47, 48] with the added advantage of having a low computational cost at the test phase. This is in general achieved using state of the art RL algorithms, namely Policy Gradient (PG), Deep Deterministic Policy Gradient (DDPG) [49] and Proximal Policy Optimization (PPO) [50].

What most of these examples do not address, however, is the robustness to uncertain spacecraft model and environment. While it is true that neural networks learn well within the training distributions, they generally fail when performing extrapolation. This may pose a risk of unstable behavior in the guidance law when visiting states that are outside the training envelope. Another major shortcoming when it comes to classical RL is sample inefficiency: many

samples are needed to learn trivial tasks, and the problem gets worse as the number of parameters and input variables increases. Reinforcement Meta-Learning (RML) was introduced [51] to address these shortcomings. Meta-Learning or *learning to learn* and its RML version, trains the policy on a distribution of environments or Markov Decision Processes (MDP) making the system adaptable to quickly learn in new situations. Consequently, the system tends to converge faster to quasi-optimal solutions. RML can play a pivotal role in space exploration. There have been a few recent publications that proved the viability of RML in space guidance applications and its superiority with respect to classical RL when uncertain environments and actuator failures are considered. For example, Gaudet et al. [52, 53] show that a guidance law based on RML achieve good performances in a six degrees-of-freedom mars landing scenario. Moreover, in [52] RML is used to create a guidance law for hovering on irregularly shaped asteroids using LIDAR sensor data. RML achieves this by training a recurrent neural network on a distribution of different environments whose parameters are randomly sampled. The resulting agent can adapt more easily to uncertain environments, leading to a much more robust guidance architecture than classical RL [54].

In this paper, we build upon those foundations by introducing optical observations. Here the goal is to employ a RML approach to devise a deep network that maps, in a closed-loop fashion, sequence of images and radar altimetry data directly into trust. The latter poses a completely new challenge: since the network input is comprised by sequences of 2D images as well as ranging observations, the closed-loop NN policy become much larger in the parameter spaces. Importantly, a novel integrated environment for photo-realistic rendering which interfaces in real-time with the learning algorithm, is developed for fast training and validation. Specifically, we created a simulator that integrates dynamics and sensor data acquisition seamlessly in a python environment, that generates accurate images using lunar digital terrain models (DTM) and a physically-based rendering engine. We leverage the python-based Blender* platform with dynamical models to create the powered descent and landing simulation environment accounting for both sensing and dynamics. The proposed RML policy integrates guidance and navigation in a single compact system that is capable of mapping sequences of observations to thrust command. This is carried out in an adaptive way within the defined distribution boundaries. Specifically, some parameters of the environment, such as the gravity acceleration and the initial spacecraft mass, are randomly sampled within some distributions, which allows the meta-learner to learn over a wide distribution of instances of a stochastic environment. Moreover, random actuator failures are introduced during training, which makes the algorithm robust to such events as well. This approach allows to obtain a complete integration of guidance and navigation in a image-based closed-loop system that is robust to perturbations and un-modeled dynamics.

The rest of the paper is structured as follows. In Section II the machine learning frameworks, specifically RL and RML, and how they are implemented in our method are described, in Section III the details about how the guidance law based on RML is implemented are presented, in Section IV the performances of the proposed algorithm for different input configurations are discussed. Finally, in Section V some final remarks are given as a conclusion statement.

*https://www.blender.org/

II. Machine Learning Frameworks

In this section, Reinforcement Learning and its extension Reinforcement Meta-Learning formulations are introduced as foundations for the learning procedure based on Proximal Policy Optimization (PPO) [50] employed in this work.

A. Reinforcement Learning

In Reinforcement Learning (RL), repeated interaction with an environment is used by an agent to learn how to complete a single or a series of tasks. In general, this environment is described as a Markov Decision Process (MDP) in which the state at a particular time depends only on the previous state. The MDP can be considered a representation of the environment with a continuous state and action space $\mathcal{S}$, $\mathcal{A}$, a state transition distribution $\mathrm{P}(x_{k+1}|x_k, u_k)$ that describes the probability of transitioning to the next state given a certain action and a reward function $r_k = \mathrm{R}(\boldsymbol{x}_k, \boldsymbol{u}_k)$, where $\boldsymbol{x} \in \mathcal{S}$, $\boldsymbol{u} \in \mathcal{A}$ and r_k is a scalar reward signal at step k. When the state cannot be observed directly, or the information is affected by noise, the MDP becomes a partially observable MDP (POMDP). In a POMDP, the state $\boldsymbol{x}$ becomes a hidden state, and the observations o are provided through an observation function $O(x)$. This is what happens in image or LIDAR based navigation where the observation embeds the information about the state without it being explicitly available to the agent. The agent operates in the environment defined by the POMDP using a parametrized policy π_θ, generating an action $\boldsymbol{u}_k$ based on the observation o_k, receiving a reward r_k and the following observation o_{k+1}. It should be noted that in this case, it is not true that each observation retains all the possible information about the system. The complete history of the sequence of observations is needed to predict the following states. The reinforcement learning algorithm task is to optimize the policy π_θ such that it maximizes the sum of the rewards collected along an episode. All path and control constraints can be included in the reward function and will be accounted for during training.

B. Reinforcement Meta-Learning

RL algorithms have demonstrated to perform well in time-dependent problems. However, because of the sample inefficiency of RL algorithms, which usually requires a vast number of trials to learn new tasks, it can be challenging to face all the uncertainties that can arise in realistic environments. On the other hand, the Reinforcement Meta-Learning (RML) algorithm is based on how human beings learn. Humans learn new tasks exploiting their previous knowledge: this concept can be summarized as "learning how to learn" through experience. Hence, to deal with the multitude of uncertainties related to the landing problem, RML is employed in this work. The main advantage that RML has over RL is that it is possible to learn over a distribution of tasks, which is more computationally efficient than learning a sequence of specific tasks from scratch.

The implementation of such concept is not straightforward. In general RL algorithms work by mapping an observation state to an action using a neural network describing a policy. Learning is then achieved using roll-outs of such policy as

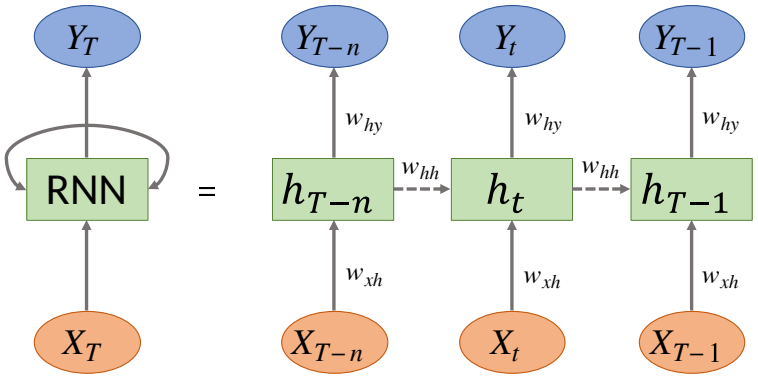


Fig. 1 RNN architecture

it interacts with a specific environment: the optimal policy is then the one that maximizes the cumulative reward:

$$\theta^* = \text{argmax}_\theta E_{\pi_\theta(\tau)}[R(\tau)] = f_{RL}(\mathbb{M}) \tag{1}$$

where θ^* are the optimal parameters of the parametrized policy π_θ, $R(\tau)$ is the cumulative reward over a trajectory τ and $\mathcal{M}$ is the MDP or POMDP. In the case of RML, the rollouts come from a distribution of tasks, which in this case are different environment conditions. This in theory allows the agent to learn how to adapt to an uncertain environment. Using the same formalization as for classical RL:

$$\theta^* = \text{argmax}_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)] \tag{2}$$

where $\phi_i = f_\theta(\mathcal{M}_i)$ is a function of the i-th MDP corresponding to the i-th task.

There are two methods to implement RML. The first is based on the traditional policy gradient algorithm: it works by training the model on a distribution of tasks obtaining a general model, and fine-tuning its parameters in the test-phase using a small number of rollouts, in what is termed Model Agnostic Meta Learning (MAML) [51]. The second approach, adopted in this paper, is based on Recurrent Neural Networks (RNN), inspired by the work of Wang et al. [55]. We synthesize the policy as a RNN, a particular NN capable of mapping temporal relationships in a sequence of inputs to the output. This is done by the inner nodes (gates) that retain information about the input data's temporal variation: this property makes RNN suitable to be used within the RML framework. Figure 1 shows a representation of an RNN. In this paper, the adopted RNN is a Gated Recurrent Unit (GRU) [56].

Indeed, for this algorithm, the policy is represented by a RNN, where its hidden states are updated through a specific task depending on the time-step, while the weights are learned across tasks using normal gradient descent. This approach allows the RNN to retain the knowledge about the tasks it has faced. Once trained, the net can eventually recognize a task by the temporal evolution of the input states and act accordingly.

The agent interacts with the environment using the action, which modifies the environment itself. Then, new observations and the reward signal are passed to the agent, and the cycles repeat until the task is completed or the constraints are violated. Initially, the agent's actions are random, which allows the agent to explore the state and action spaces and gather information about the environment and which action is to be preferred, given a particular observation. The information about the goodness of an action is embedded in the reward signal. As the learning progresses, the exploration is reduced in favor of the exploitation of the knowledge of the environment. For most applications (landing guidance is one of them), the policy is deployed as a deterministic law in which exploration is switched off.

C. Learning Procedure

In this section, the learning process is described. Note that this procedure is common for both RL and MRL. Let $\mathbf{x}_k$ be the observation, coming from either MDP or POMDP, provided by the environment to the agent at time-step k. Each episode results in a sequence of observations which we will call trajectory. A step in each trajectory at time t_k can be represented as $(o_k, \boldsymbol{u}_k, r_k)$ where o_k and r_k are the observation and the reward returned by the environment and $\mathbf{u}_k$ is the action taken. The reward can be a function of both the observation and the action. The reward is then typically discounted the further it is in the future. This is done to put a finite horizon to the task and facilitate temporal credit assignment. Then the sum of discounted rewards for a trajectory can be defined as the return:

$$r(\tau) = \sum_{i=0}^{T} \gamma^i r_i \tag{3}$$

where $\tau = [\mathbf{x}_0, \mathbf{u}_0, \ldots, \mathbf{x}_T, \mathbf{u}_T]$ denotes the trajectory, $\gamma \in [0, 1)$ is the discount factor and r_i is the reward at step i. The objective function that RL methods seek to optimize is given by:

$$J(\theta) = \mathbb{E}_{p(\tau)}[r(\tau)] = \int_{\mathbb{T}} r(\tau) p_\theta(\tau) d\tau \tag{4}$$

where:

$$p_\theta(\tau) = \left[\prod_{k=0}^{T} p(\mathbf{x}_{k+1} | \mathbf{x}_k, \mathbf{u}_k) \right] p(\mathbf{x}_0) \tag{5}$$

where $\mathbb{E}_{p(\tau)}$ denotes the expectation of the reward over the trajectories. Now if we consider the action $\mathbf{u}_k$ as a stochastic function of θ, or $\mathbf{u}_k \sim \pi_\theta(\mathbf{u}_k | \mathbf{x}_k)$, then the policy gradient expression becomes:

$$\nabla_\theta J(\theta) = \int_{\mathbb{T}} \sum_{k=0}^{T} r_k(\mathbf{x}_k, \mathbf{u}_k) \nabla_\theta log \pi_\theta(\mathbf{u}_k | \mathbf{x}_k) p_\theta(\tau) d\tau \approx \sum_{i=0}^{M} \sum_{k=0}^{T} r_k(\mathbf{x}_k^i, \mathbf{u}_k^i) \nabla_\theta log \pi_\theta(\mathbf{u}_k^i | \mathbf{x}_k^i) \tag{6}$$

where the integral over the trajectory τ is approximated using the monte-carlo rolluts samples $\tau^i \sim p_\theta(\tau)$, given the environment's transition probabilities, $p(\mathbf{x}_{k+1} | \mathbf{x}_k)$, which in this case are the deterministic equations of motion. The

expression in 6 is the policy gradient equation and is the basic concept on which the REINFORCE [57] algorithm is based. In general this is used as a baseline as this was later improved through the years. In particular it was shown that, instead of using the actual reward $r_k(\mathbf{x}_k, \mathbf{u}_k)$, one can use the action-value function $Q^\pi(\mathbf{x}_k, \mathbf{u}_k)$ called normally Q-function. Moreover, to reduce the variance of the policy gradient, a state dependent basis can be subtracted from $Q^\pi(\mathbf{x}_k, \mathbf{u}_k)$. This basis is normally called value function $V^\pi(\mathbf{x}_k)$ and the quantity that is then used to approximate the policy gradient is the advantage function $A^\pi(\mathbf{x}_k, \mathbf{u}_k) = Q^\pi(\mathbf{x}_k, \mathbf{u}_k) - V^\pi(\mathbf{x}_k)$. This value function can be approximated using the rollouts by training a second neural net which is normally referred to as critic. This method is known as Advantage-Actor-Critic (A2C) Method [58] and the policy gradient using this method becomes:

$$\nabla_\theta J(\theta) = \sum_{i=0}^{M} \sum_{k=0}^{T} \nabla_\theta log \pi_\theta(\mathbf{u}_k^i | \mathbf{x}_k^i) A_w^\pi(\mathbf{x}_k^i, \mathbf{u}_k^i) \tag{7}$$

Once the gradient is calculated, it is used to update the policy by simply moving in its direction:

$$\theta^+ = \theta^- + \beta_\theta \nabla_\theta J(\theta)|_{\theta=\theta^-} \tag{8}$$

where $\beta_\theta > 0$ is the learning rate.

D. Proximal Policy Optimization

What we used to optimize the policy in this case is a derivation of the A2C method. The PPO approach [50] belongs to the family of policy gradient algorithms and has demonstrated state-of-the-art performances on many benchmark RL problems. It is developed as a derivation of the Thrust Region Policy Optimization (TRPO) Method [59]. This method formulates the policy optimization problem in a way such that the size of the gradient step taken during each iteration is restricted using a dynamically calculated constraint. The TRPO policy update problem is formulated as:

$$\begin{aligned} \min_\theta \quad & \mathbb{E}_{p(\tau)} \left[\frac{\pi_\theta(\mathbf{u}_k|\mathbf{x}_k)}{\pi_{\theta_{old}}(\mathbf{u}_k|\mathbf{x}_k)} A_w^\pi(\mathbf{x}_k, \mathbf{u}_k) \right] \\ s.t. \quad & \mathbb{E}_{p(\tau)} [K_L(\pi_\theta(\mathbf{u}_k|\mathbf{x}_k), \pi_{\theta_{old}}(\mathbf{u}_k|\mathbf{x}_k))] \leq \delta \end{aligned} \tag{9}$$

where K_L is the Kullback-Leibler divergence [60] between the present and the old policy. The parameter δ is a tuning parameter that imposes a bound on the update. It is proven that if the update is bounded at each iteration by a parameter $C(K_L)$, the policy improves monotonically towards the optimal. This in general leads to prohibitively small update so Equation 9 with a constant constraint parameter is used instead. Additionally, Equation 9 is approximately solved using the conjugate gradient algorithm, which approximates the constrained optimization problem given by Equation 9 with a linearized objective function and a quadratic approximation for the constraint. The PPO method approximates the TRPO optimization process by accounting for the constraint on the policy update with a clipped objective function. This

can be expressed in terms of the probability ratio $p_k(\theta)$ given by,

$$p_k(\theta) = \frac{\pi_\theta(\mathbf{u}_k|\mathbf{x}_k)}{\pi_{\theta_{old}}(\mathbf{u}_k|\mathbf{x}_k)} \tag{10}$$

The objective function is then:

$$L(\theta) = \mathbb{E}_{p(\tau)}\left[\min[p_k(\theta), \text{clip}(p_k(\theta), 1-\epsilon, 1+\epsilon)]A^\pi_w(\mathbf{x}_k, \mathbf{u}_k)\right] \tag{11}$$

As stated above, we use an approximation of the advantage function that is the difference between the empirical return (discounted reward) and a state value function baseline and gives information about how much better an action is with respect to the average action:

$$A^\pi_w(\mathbf{x}_k, \mathbf{u}_k) = \left[\sum_{l=k}^{T} \gamma^{l-k} r(\mathbf{u}_l, \mathbf{x}_l)\right] - V^\pi_w(\mathbf{x}_k) \tag{12}$$

Where $\gamma \in [0, 1)$ is the discount factor and is closer to one the more the algorithm should care about rewards collected far into the future. Note that the subscript w was added to underline the fact that the advantage function depends on the approximation of the value function performed by the critic. The value function $V^\pi_w(\mathbf{x}_k)$ is learned using the cost function:

$$L(w) = \frac{1}{2M}\sum_{i=1}^{M} \max\left[L_1, L_2\right]^2 \tag{13}$$

where L_1 and L_2 are the normal and clipped value function as per OpenAI baselines [61]:

$$L_1 = V^\pi_w(\mathbf{x}^i_k) - \left[\sum_{l=k}^{T} \gamma^{l-k} r(\mathbf{u}_l, \mathbf{x}_l)\right] \tag{14}$$

$$L_2 = \left[V^\pi_{w,old} + clip\left(V^\pi_w - V^\pi_{w,old}, -\epsilon, \epsilon\right)\right] - \left[\sum_{l=k}^{T} \gamma^{l-k} r(\mathbf{u}_l, \mathbf{x}_l)\right] \tag{15}$$

and M is the number of rollout trajectories. In practice, policy gradient algorithms update the policy using a batch of trajectories (roll-outs) collected through interaction with the environment. Each trajectory is associated with a single episode, with a sample from a trajectory collected at step k consisting of observation $\mathbf{x}_k$, action $\mathbf{u}_k$, and reward $r_k(\mathbf{x}_k, \mathbf{u}_k)$. Finally, gradient ascent is performed on θ and gradient decent on w. The update equations are:

$$w^+ = w^- - \beta_w \nabla_w L(w)|_{w=w^-} \tag{16}$$

$$\theta^+ = \theta^- + \beta_\theta \nabla_\theta J(\theta)|_{\theta=\theta^-} \tag{17}$$

where β_w and β_θ are the learning rates for the value function, V^π_w, and policy, $\pi_\theta(\mathbf{u}_k|\mathbf{x}_k)$, respectively. The policy and value function are learned concurrently. The exploratory action distribution is a Gaussian distribution with mean

$\pi_\theta(\mathbf{x}_k)$ and a diagonal covariance matrix. Because the log probabilities are calculated using the exploration variance, the degree of exploration automatically adapts during learning so that the objective function is maximized. It should be mentioned that the parameters θ are dependent on the neural network used. In this work the neural network is a recurrent neural network (RNN), which, following the work by Wang et al. [55] , enables the agent to achieve meta-learning.

The pseudo-code training algorithm can be seen in Algorithm 1.

Algorithm 1 Meta-RL - Recurrent network

procedure WHILE TRAINING
  for i in tasks **do**
    initialize hidden state $\mathbf{h}_0$
    for t in time-steps **do**
      sample an action from π_{h_t}
      obtain next observation s_{t+1} and reward r_t
      update policy hidden state $\mathbf{h}_{t+1} = f_\theta(\mathbf{h}_t, s_t, a_t, s_{t+1}, r_t)$
  compute advantage $A^\pi_w(\mathbf{x}_k, \mathbf{u}_k) = \left[\sum_{l=k}^{T} \gamma^{l-k} r(\mathbf{u}_l, \mathbf{x}_l)\right] - V^\pi_w(\mathbf{x}_k)$
  for epoch in epochs **do**
    for minibatch in minibatches **do**
      update value function parameters $w^+ = w^- - \beta_w \nabla_w L(w)|_{w=w^-}$ using loss in Eq. 14
      compute clipped advantage $L(\theta) = \mathbb{E}_{p(\tau)}\left[\min[p_k(\theta), \text{clip}(p_k(\theta), 1-\epsilon, 1+\epsilon)] A^\pi_w(\mathbf{x}_k, \mathbf{u}_k)\right]$
      update policy parameters $\theta^+ = \theta^- + \beta_\theta \nabla_\theta J(\theta)|_{\theta=\theta^-}$

III. Reinforcement Meta-Learning Guidance Algorithm Implementation

A. Environment and Sensor Specification: Blender Environment for Reinforcement Learning (BERL)

1. Equations of motion

In this paper, the 3 Degrees of Freedom (3DOF) lunar soft landing problem is considered. The lander's attitude is not taken into account, whereas the translational motion is described in an ortho-normal inertial reference frame centered on the nominal landing target (here simply referred to as *I* frame). The z-axis of the *I* frame is supposed to point upwards. Hence, the equations of motion are:

$$\ddot{\mathbf{r}} = \mathbf{g} + \frac{\mathbf{T}}{m} \tag{18}$$

$$\dot{m} = -\frac{\|\mathbf{T}\|}{I_{sp} g_0} \tag{19}$$

where $\mathbf{r} = [r_x\ ,\ r_y\ ,\ r_z]^T$ is the state in *I* frame, $\mathbf{g}$ is the gravity vector, in this case supposed to be constant since a flat surface is considered, and $\mathbf{T}$ is the thrust vector in the same *I* frame $\boldsymbol{T} = [T_x\ ,\ T_y\ ,\ T_z]^T$ Note that the lander is equipped with throttable thrusters, capable of providing a variable thrust in all three directions. The guidance problem is the following: given the initial position and velocity, find the thrust commands to reach the desired final position and

velocity while fulfilling both path and control constraints. Thus, the boundary conditions are

$$\begin{aligned} \mathbf{r}(0) &= \mathbf{r}_0 \\ \mathbf{v}(0) &= \dot{\mathbf{r}}(0) = \mathbf{v}_0 \\ \mathbf{r}(t_f) &= \mathbf{r}_f \\ \mathbf{v}(t_f) &= \dot{\mathbf{r}}(t_f) = \mathbf{v}_f \end{aligned} \tag{20}$$

where $\mathbf{r}_0$, $\mathbf{v}_0$, $\mathbf{r}_f$ and $\mathbf{v}_f$ represent the initial and final position and velocity vectors respectively. As mentioned before, a path constraint is taken into account, and it consists of a fictitious flat surface, located at a certain target altitude, that cannot be overcome by the lander. Also, thrust constraints are given by:

$$0 < T_{min} < \|\mathbf{T}\| < T_{max} \tag{21}$$

where T_{min} and T_{max} represent the minimum and maximum thrust provided by the thrusters.

The initial mass and the gravity vector are varied among the episodes to consider a more general environment for the reinforcement meta-learning algorithm. Moreover, possible engine failures are also considered randomly at the initial time instant of each episode. This is done by reducing the computed commanded thrust along the z-axis and along one of the other two axes using a scale factor.

2. Blender Environment for Reinforcement Learning (BERL)

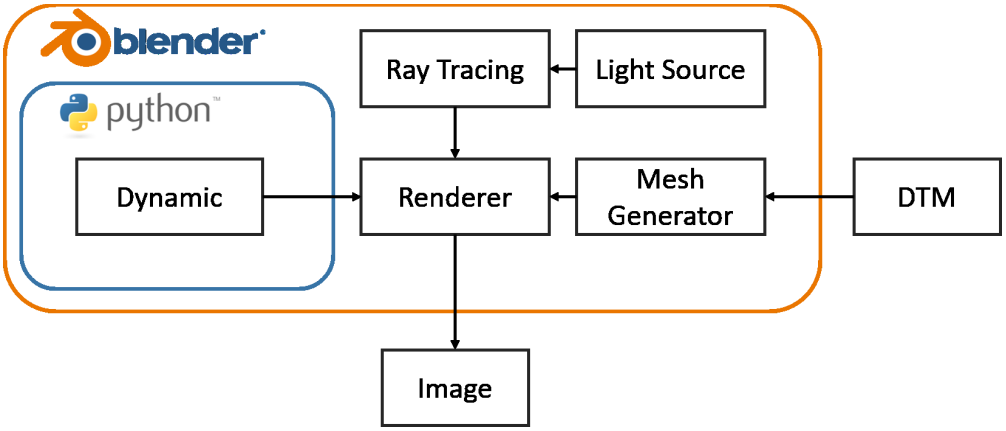
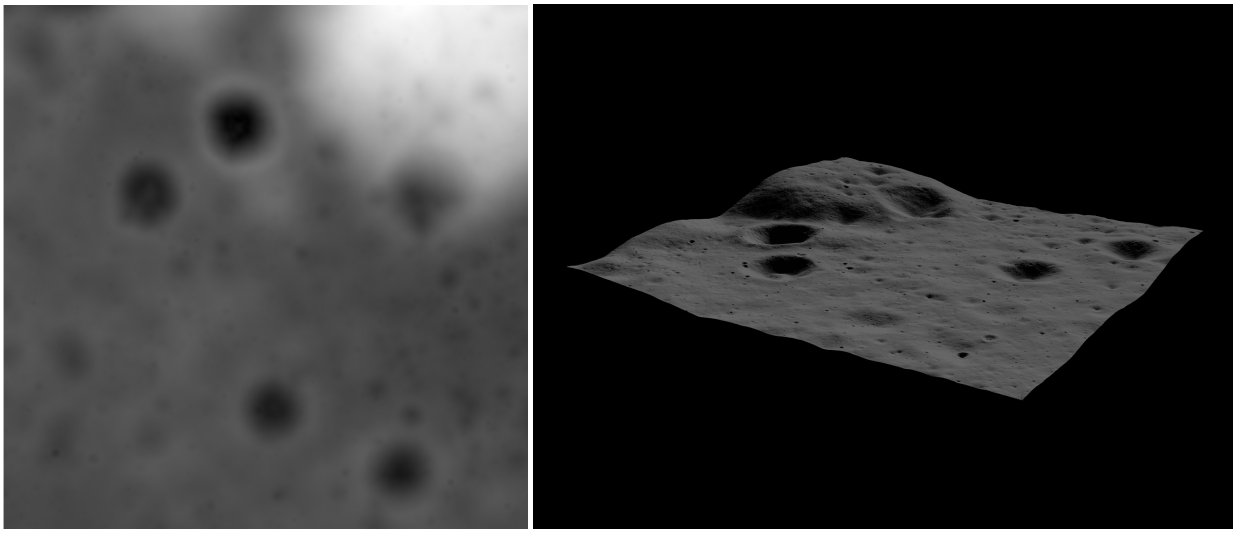


Fig. 2 Blender environment for reinforcement learning

The spacecraft is assumed to have various devices, such as (but not limited to) radars and optical sensors, that can provide a multitude of information inherent to its position and velocity. Hence, the policy is continuously fed with this information. The optical sensor's boresight is always aligned with the negative direction of the z-axis of the *I*

frame; thus, it is always pointed towards the ground. The images coming from the optical sensor are generated within a simulated environment built in the open-source software Blender. Specifically, the DTM representing the Apollo 16 landing site, taken from the Lunar Reconnaissance Orbiter (LROC) database[†], is imported into Blender and it is used to generate a realistic ground model. Images are then generated using a renderer within Blender by simulating a camera moving in such an environment. In particular, we use Cycles, a physically-based ray tracer that computes light traces from all the light sources in the scene and their interactions with the objects. This allows simulating the parallel light rays' interaction coming from the simulated Sun with the lunar surface material, which is generally characterized by high roughness and low reflectivity. Figure 3 shows the DTM and a rendered view of the ground model using the raytracer.



(a) DTM (b) Rendered surface model

Fig. 3 Apollo 16 landing site

The DTM is a 32 bit grayscale image with a resolution of 3000x3000 pixels. Blender uses it as a displacement map, where each pixel's value represents the elevation of the terrain at that point. This information is exploited by the renderer to compute the light bounces of the light emitted by the Sun. Moreover, the simulated camera provides high fidelity observations with respect to real scenarios, where new features appear as the altitude decreases. This is obtained by increasing the polygon count of the simulated ground as the camera approaches the surface. This technique also allows for faster rendering times when the camera is far away from the ground, i.e., in the first part of each trajectory. Since the reinforcement meta-learning framework is coded in Python, the choice of Blender results to be very convenient as it has a built-in Python interpreter. Thus, the learning algorithm can be run directly in the Blender environment, which allows the image observations to be fed directly as a pixel data array to the RML agent (see below). One can note that the simulator described above can be easily adapted to other frameworks, such as Mars and asteroid landing and close-proximity and docking operations. The possibilities are only limited by the availability of detailed 3D models and

[†]http://wms.lroc.asu.edu/lroc

textures, It should, however, be noted that, even if real 3D models and textures are not available, the simulator can be used to generate realistic scenarios using the advanced features of 3D modelling and texturing available within Blender.

B. Observed information

In the case of this paper, the environment contains the model of the lander, the equations of motion, and all the physical constraints of the problem. One should note that, once the training of the net has been carried out, only the policy is deployed, while the value function approximator (critic) is not. This means that it is possible to feed the critic with any information that might be useful to improve the learning performance during training. Thus, the following information are passed to the critic: the velocity error between the true velocity and the reference velocity $\mathbf{v_{err}} = \|\mathbf{v} - \mathbf{v}_{targ}\|$, the time-to-go t_{go}, the lander state ($\mathbf{r}$) and velocity ($\mathbf{v}$) vectors.

$$o_{VF} = [\mathbf{v_{err}} \quad t_{go} \quad r_x \quad r_y \quad r_z \quad \dot{r}_x \quad \dot{r}_y \quad \dot{r}_z] \tag{22}$$

On the other hand, the policy has access to a limited amount of information. In this work, we analyze three different scenarios according to the information that is passed to the policy. Let $\mathbf{I}$ be a 32x32 array representing the raw grayscale pixel data from the raytracer. The information used for the three scenarios are:

1) In the first scenario, the image acquired by the optical sensor, the vertical position and velocity are employed.

$$o_{\pi_\theta} = [\mathbf{I} \quad r_z \quad \dot{r}_z] \tag{23}$$

2) In the second scenario, all the information of the previous case are considered along with the velocity horizontal components.

$$o_{\pi_\theta} = [\mathbf{I} \quad r_z \quad \dot{r}_x \quad \dot{r}_y \quad \dot{r}_z] \tag{24}$$

3) In the third scenario, the image is not employed and the policy fed with all the components of position and velocity vectors.

$$o_{\pi_\theta} = [r_x \quad r_y \quad r_z \quad \dot{r}_x \quad \dot{r}_y \quad \dot{r}_z] \tag{25}$$

We analyzed different input scenarios to test the method with varying amounts of available information depending on the onboard sensors. In particular, if just a camera and a radar altimeter are available, the first scenario is the one that should be considered. Indeed, the same sensors could also be used for the second scenario if the horizontsal velocity is estimated from the images, for example, through optical flow analysis [62, 63]. As can be easily understood, the last scenario is the one that provides the most accurate navigation since more information are available and a more precise guidance is to be expected. However, at the same time, it is unlikely that the whole state is available for the lunar

landing problem. The action space is continuous and corresponds to $\mathbb{R}^3$ as we consider motion in a 3DOF problem. A schematic representation of the whole system can be seen in Fig. 4. To further justify the use of recurrent networks to

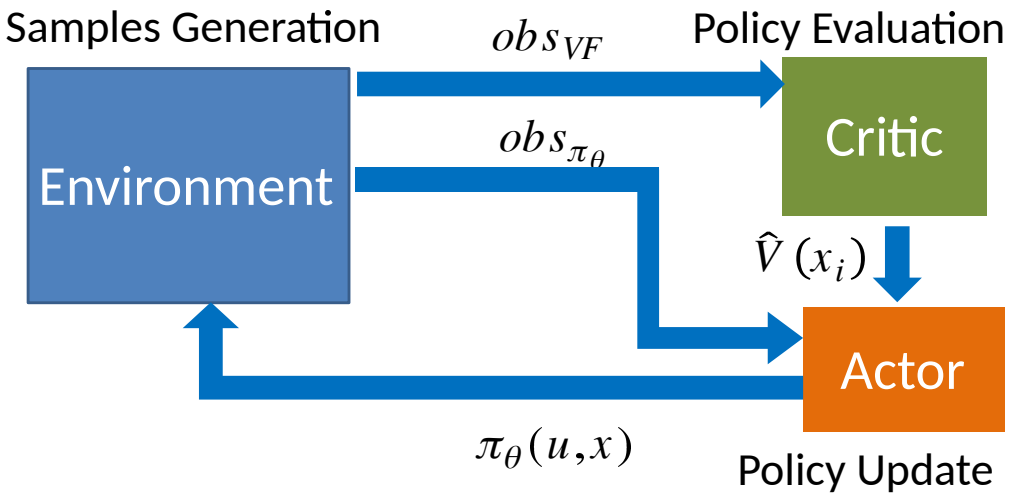


Fig. 4 Actor-Critic framework

solve the reinforcement meta-learning problem, we used the hidden states to replace the missing observations that are not available at the beginning of each trajectory. This is done during interaction with the environment by returning both the observation-action-reward vector and the policy's hidden state. The forward pass through the policy then works by unrolling the batch of inputs before the recurrent layer, reshaping it according to the layer dimension and recurrent steps (in this case, we consider 100 steps).

C. Reward Function

The most intuitive way to reward the agent in a soft landing task is to give a positive reward if the agent performs a successful landing (e.g. arrives close to the target with low speed) and penalize it if it violates a constraint. These kinds of rewards are considered sparse because the agent is rewarded or penalized only at specific time-steps during a rollout. It is known that reinforcement learning does not perform well with sparse rewards [64]. This is due to the fact that it is improbable that a successful landing is observed using a randomized exploratory policy. To solve this problem, we employed a distributed reward function, partially inspired by previous works of the authors [37, 52–54]. The reward at each time-step is computed as the error between the current velocity vector and a reference velocity. Specifically, the reference velocity is based on a gaze heuristics potential function that is aligned with the line of sight between the lander and the target at all times. The target position and velocity are indicated as $\boldsymbol{r}_t$ and $\boldsymbol{v}_t$ respectively. This ensure pinpoint but not necessarily soft landing. To ensure that the final velocity is minimized, the agent estimates the time-to-go as the ratio of the range and the magnitude of the lander's velocity. It then reduces the targeted velocity as time-to-go decreases. This is incentivized by giving an additional positive reward at the end of successful trajectories. This reward

Table 1 Hyperparameters

τ_1 (s)	τ_2 (s)	α	η	δz (m)
20	100	-0.01	0.01	20

increases in finite increments the closer the final state is to the target state. The reward function has the following form:

$$r = \alpha\|\mathbf{v} - \mathbf{v}_{targ}\| + \eta + \kappa(r_{lim,1} < \|\mathbf{r}(t_f) - \mathbf{r}_t\| < r_{lim,2}) + \xi(v_{lim,1} < \|\mathbf{v}(t_f) - \mathbf{v}_t\| < v_{lim,2}) \tag{26}$$

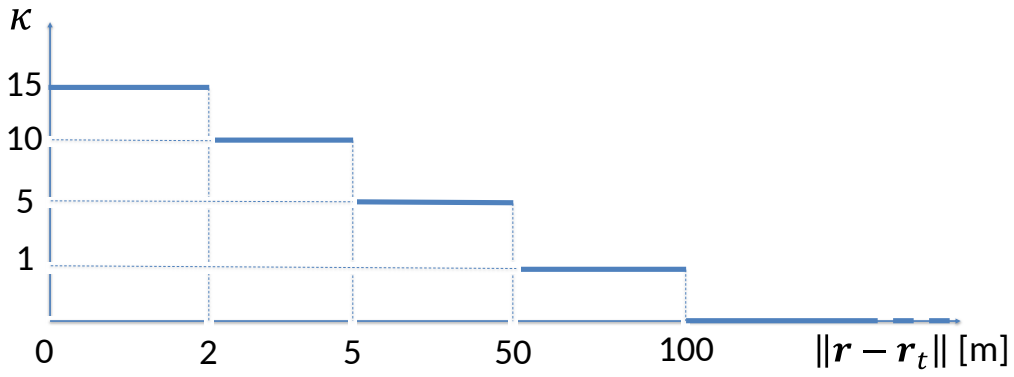
Where:

$$\begin{aligned}
\mathbf{v}_{targ} &= -v_0 \left(\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|}\right)\left[1 - \exp\left(-\frac{t_{go}}{\tau}\right)\right] \\
t_{go} &= \frac{\|\hat{\mathbf{r}}\|}{\|\hat{\mathbf{v}}\|} \\
\hat{\mathbf{r}} &= \begin{cases} \mathbf{r} - \mathbf{r}_{t'} & \text{if} \quad r_z > r_{t',z} \\ \mathbf{r} - \mathbf{r}_t & \text{otherwise} \end{cases} \\
\hat{\mathbf{v}} &= \begin{cases} \mathbf{v} - \mathbf{v}_{t'} & \text{if} \quad r_z > r_{t',z} \\ \mathbf{v} - \mathbf{v}_t & \text{otherwise} \end{cases} \\
\tau &= \begin{cases} \tau_1 & \text{if} \quad r_z > r_{t',z} \\ \tau_2 & \text{otherwise} \end{cases}
\end{aligned} \tag{27}$$

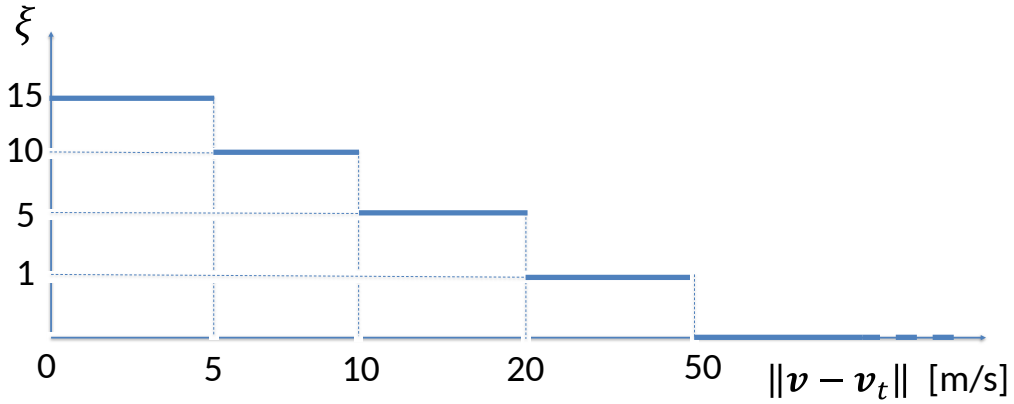
with $\mathbf{r}_{t'} = \mathbf{r}_t + [0, 0, \delta z]$ is an intermediate position above the desired final position and:

- α is a negative term that penalizes the error with respect to the target velocity
- η is a positive constant that encourages the agent to perform more steps avoiding collisions with the constraints
- κ and ξ are piecewise bonuses given if the final position and velocity are within certain intervals represented by $[r_{lim,1}, r_{lim,2}]$ and $[v_{lim,1}, v_{lim,2}]$, respectively. The boundaries intervals and the corresponding κ and ξ values can be easily deduced from the plots in Fig. 5. One can note that κ and ξ are both equal to 0 if $||\mathbf{r} - \mathbf{r}_t||$ and $||\mathbf{v} - \mathbf{v}_t||$ are respectively greater than 100 m and 50 m/s. Moreover, the third and fourth terms of the right hand side of Eq. (26) are evaluated only at the final time-step.
- v_0 is the norm of the initial velocity along a specific rollout trajectory.

A quasi-vertical motion is obtained during the last phase of the landing trajectory by following this reward shaping function.



(a) κ



(b) ξ

Fig. 5 Parameters κ and ξ vs position and velocity intervals

IV. Training and Testing Results

In this section, test results are presented for each of the three training scenarios previously explained. For all the cases, the initial position and velocity vectors' components are sampled randomly from uniform distributions, whose bounds are reported in Table 2. The nominal initial mass is equal to 1500 kg. The final state is always the same for all the episodes and it is equal to [0, 0, 200] m for the position and [0, 0, -1] m/s for the velocity. To make the guidance algorithm more robust to uncertain environments, the initial mass and z component of the gravity vectors are sampled randomly within the range $[0.9m_0, 1.1m_0]$ and $[0.9g_z, 1.1g_z]$, respectively. Instead, the gravity vectors' x and y components are sampled randomly within the range $[-0.1g_z, 0.1g_z]$. Moreover, random engine failures might manifest themselves as reduced thrust along the z-axis and along one of the other two axes. Indeed, the commanded thrust is 2/3 of the thrust estimated by the NN. For the three scenarios, the value function is approximated by the NN whose architecture is shown in Fig. 6, where FC stands for Fully Connected layer. The rest of this section will present in detail the three test scenarios and the corresponding results.

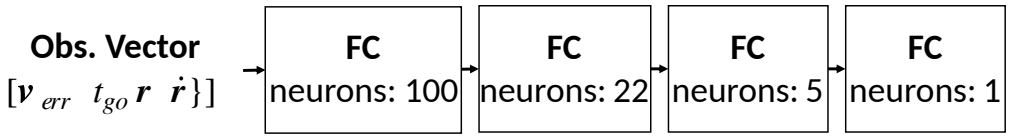


Fig. 6 Critic network

Table 2 Initial conditions distribution bounds

	Position		Velocity	
	min (m)	max (m)	min (m/s)	max (m/s)
X	-200	200	-20	-20
Y	-1500	-1000	50	70
Z	1500	1800	-40	-20

A. Case 1: images + vertical position + vertical velocity

In this case, the NN is fed with a sequence of images, the spacecraft's vertical position, and vertical velocity. The NN is designed accordingly using a mixed input architecture. In particular, a CNN, which handles the images, is paired with a Fully Connected (FC) head that processes the vector observations. The output of the mixed layers is then fed into a GRU. Finally, two FC layers complete the NN. The proposed architecture is shown in Fig. 7 along with the chosen hyperparameters. The results are reported in Fig. 8. As can be seen from the reward, it starts with a negative value at the beginning of the training since the spacecraft does not reach the prescribed tolerances at the end of the trajectory and the only non-null terms in the reward function (Eq. 26) are the negative term related to the velocity error and the small positive incentive to perform more steps (η). The effect of η is clearly visible in Fig. 8(a), where the increase of the reward is strictly related to an increasing number of time steps per trajectory. This curve (purple dotted line) reaches a plateau, while the reward continues to increase due to the decreasing velocity error along the trajectory and the last two terms corresponding to the final state error. Therefore, the final value of the reward, achieved after 20000 episodes, is completely positive, which means that a successful trajectory is obtained. Figure 8(b) shows the 100 trajectories of the Monte-Carlo simulation, whereas Fig. 8(c) reports the distribution of the final position and velocity errors, and the corresponding statistics are shown in Table 3. In particular, the position error in the x-y plane is below 5 m (the z component is not shown because it is always achieved since the episodes are interrupted when the spacecraft reaches an altitude of 200 m). The x and y components of the final velocity vector are globally below 2 m/s. The final z component of the velocity is shown in the last histogram, where it is possible to note that for most of the trajectories, it is about -2 m/s (one should remember that the target velocity is [0,0,-1] m/s). For case 1, 1-σ, 2-σ and 3-σ error ellipses for the final x and y components of position and velocity are shown in Fig. 8(c).

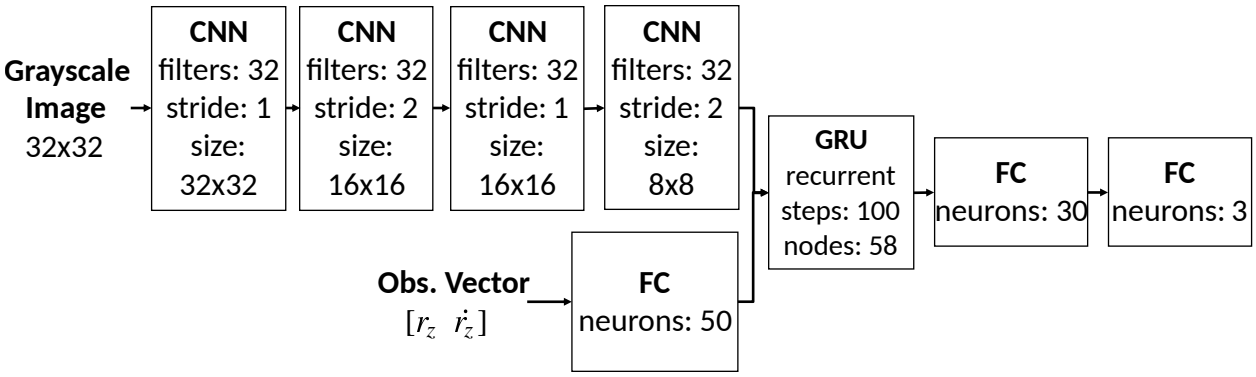


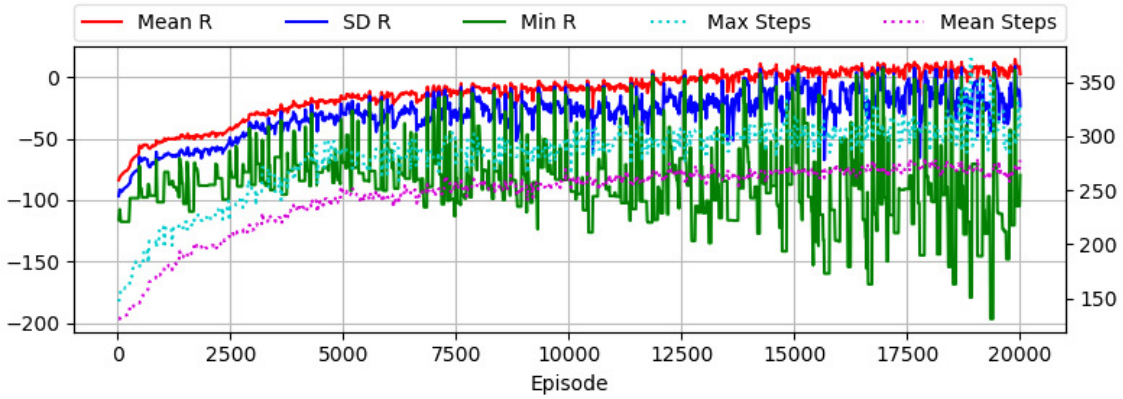
Fig. 7 Network architecture for case 1.

B. Case 2: images + vertical position + velocity vector

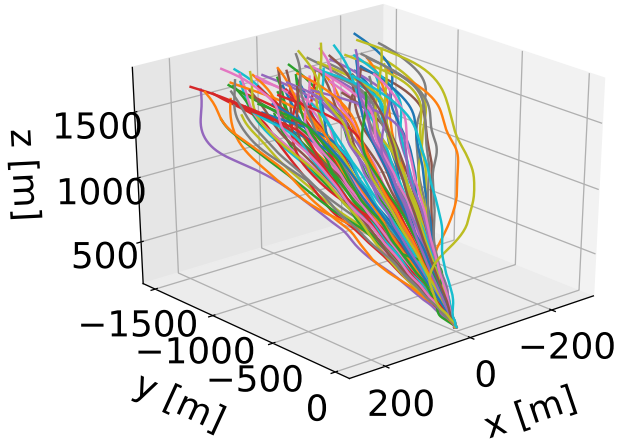
In the second case, the NN is fed with a sequence of images, vertical position, and the full velocity vector. Even in this case, the NN is designed using a mixed input architecture. The same architecture of test 1 is used, but with different hyperparameters. The proposed architecture is shown in Fig. 9 along with the chosen associated hyperparameters. The results are presented in Fig. 10. The same considerations about the reward discussed for the previous case are still valid for Case 2. It is possible to see from Fig. 10(c) and Table 3 that the position and velocity error ellipses have lower semi-major axes values with respect to Case 1, which means that accuracy and precision are higher. This was expected since additional information (the entire velocity vector) is passed to the policy network. However, the uncertainty on the z component of the velocity is slightly higher than in the previous case.

C. Case 3: velocity error + time-to-go + altitude

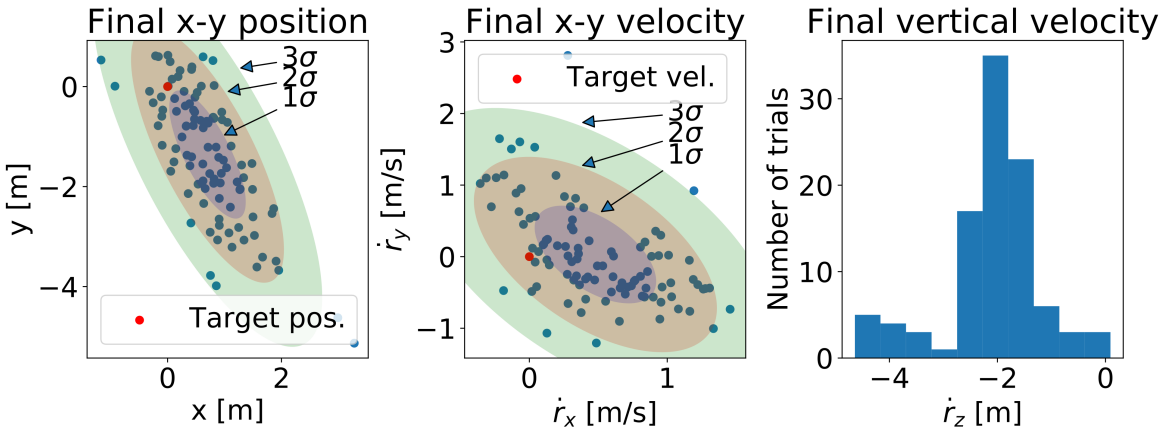
In this case, the NN is fed with the velocity error between the current velocity and the target $\mathbf{v}_{targ}$ originating from the velocity field as explained in Section III, along with the time-to-go t_{go} and the altitude. The network architecture is shown in Fig. 11. In this case the agent reaches a high level of precision as shown by the error ellipses in Fig. 12 and Table 3, with standard deviation both for position and velocity below 0.2 m and m/s respectively. On the other hand, the relative accuracy of the solution is lacking in comparison, with the target being only contained within the 3σ ellipse for the velocity and being completely out of the 3σ ellipse for the position. This difference between high precision and lower accuracy could be due to the way the episodes are terminated, which favors the vertical position accuracy with respect to the x and y components of the position and velocity vectors. Indeed, the episode is terminated when the altitude falls below 200 meters, no matter what the values of the other position and velocity components are. Although this behaviour should be considered, it should be noted that in an absolute scale, the accuracy is still better than all the previous cases as shown by the mean of the distributions. Overall, in this case it is clear that the added information provided by the velocity error as the agent "sees" it in the reward function, allows it to learn to perform better in such environment, and consequently produce more accurate results.



(a) Reward



(b) Test trajectories



(c) Test statistics

Fig. 8 Results for Case 1.

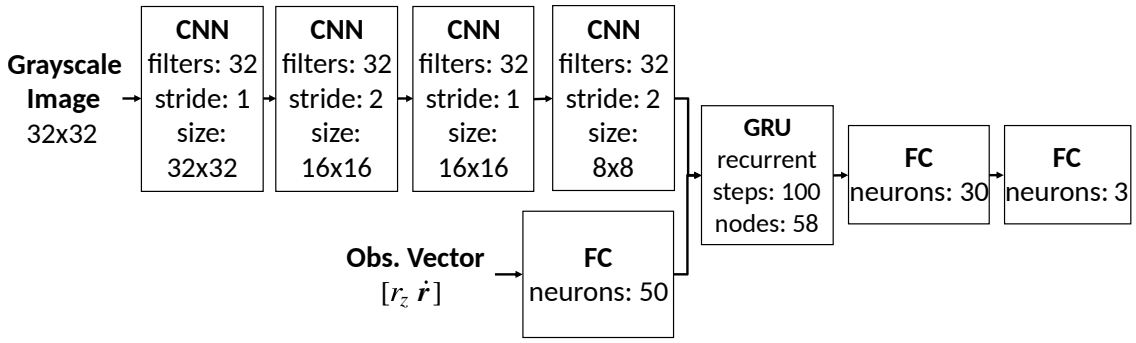
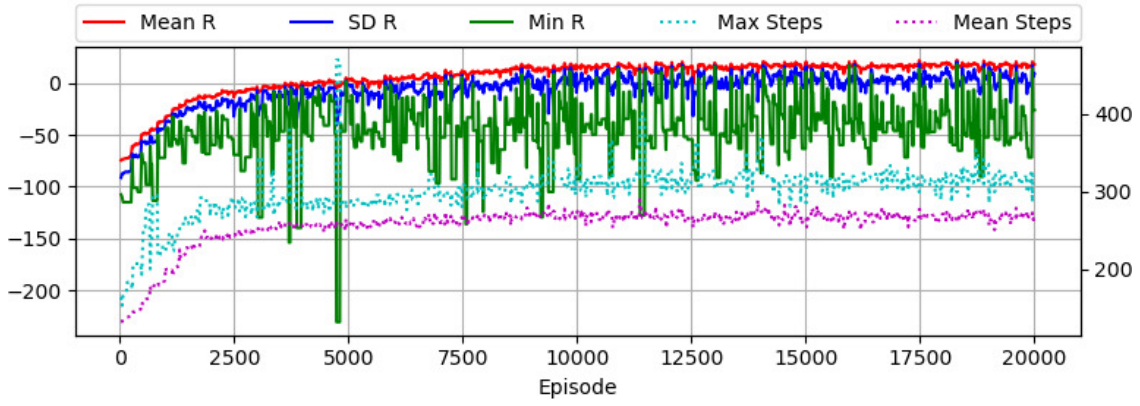


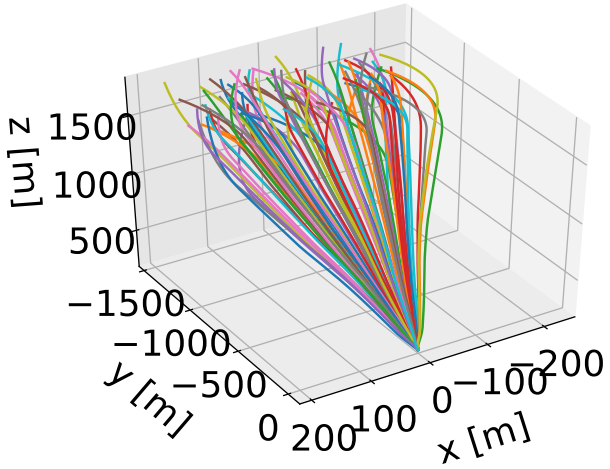
Fig. 9 Network architecture for case 2.

Table 3 Performance statistics for all cases

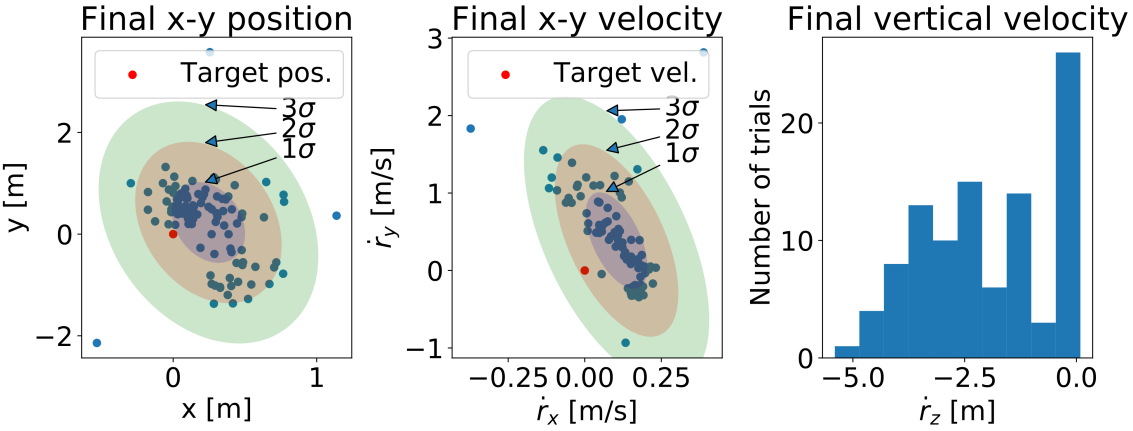
	Terminal position error (m)				Terminal velocity error (m/s)						Fuel (kg)
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	μ_z	σ_x	σ_y	σ_z	μ
Case 1	0.701	-1.358	0.668	1.275	0.479	0.030	-1.116	0.438	0.679	0.945	123.6
Case 2	0.246	0.229	0.254	0.789	0.104	0.387	-0.976	0.100	0.618	1.520	111.8
Case 3	-0.052	-0.041	0.025	0.042	0.091	-0.142	0.367	0.0413	0.107	0.176	115.9



(a) Reward



(b) Test trajectories



(c) Test statistics

Fig. 10 Results for Case 2.

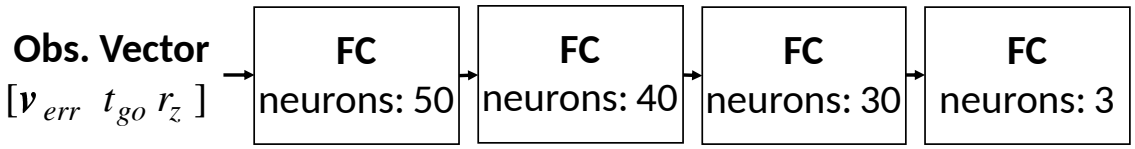
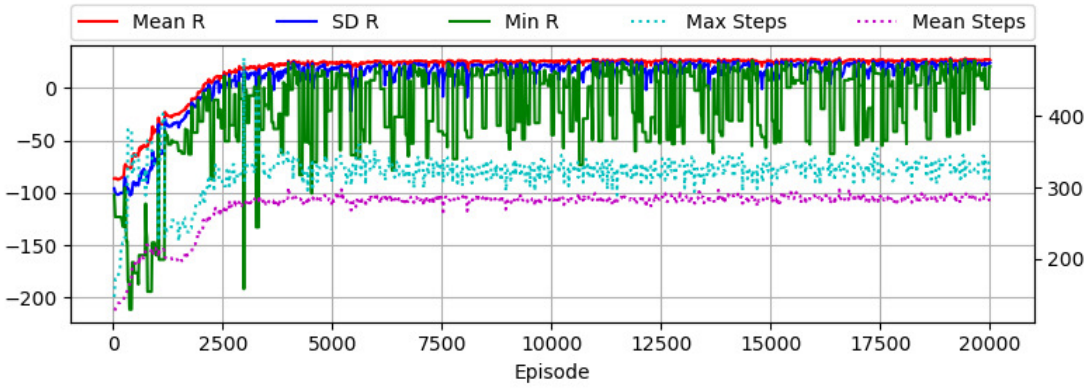


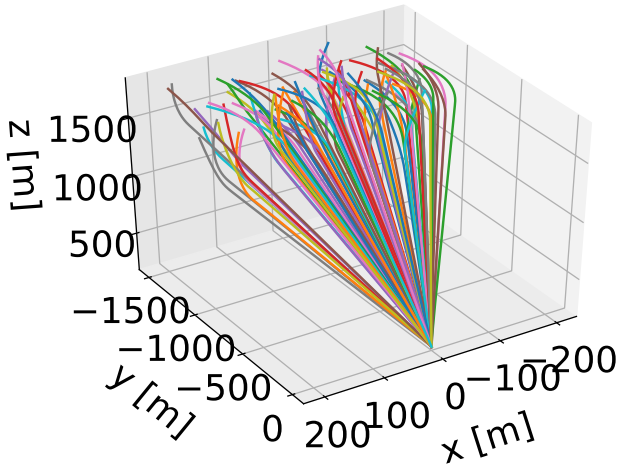
Fig. 11 Network architecture for case 3.

V. Conclusion

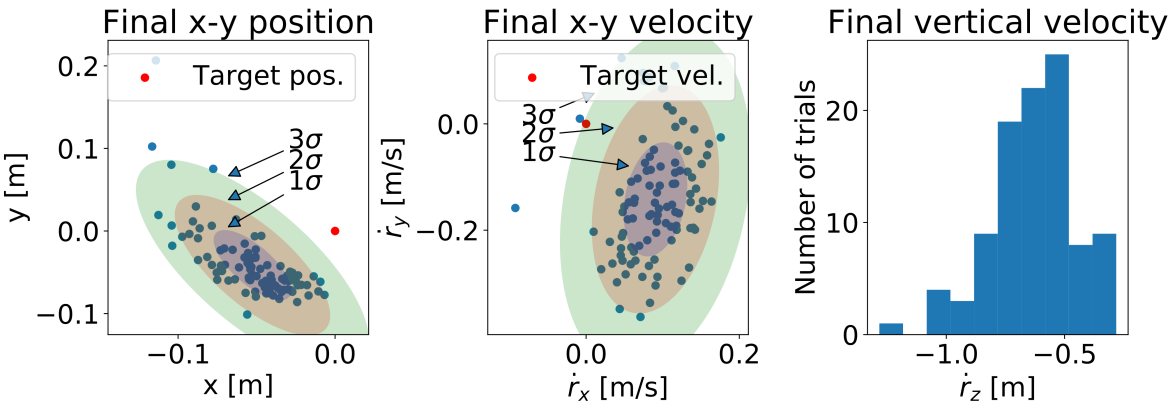
In this paper, RML is employed to obtain a closed-loop lunar landing guidance based directly on sensors data, avoiding the need for a state estimation system. A deep recurrent convolutional neural network is devised to acquire a sequence of images (coming from the optical camera) and ranging data (coming from the radar altimetry) and generate a thrust command (sensor-to-thrust). The deep policy network is designed and trained on a distribution of MDPs to enable quick adaption and learning under uncertain conditions. The method allows for increased robustness against dynamics uncertainties, such as possible perturbations of the environment, and engine failures, with respect to classical RL. This is practically obtained by using a GRU layer in the deep architecture, which enables retaining information about previous experience (i.e. environments) and allows the agent to train accurately and efficiently. Although powerful, this method relies on staying within the environment distribution used during the training, therefore the performance of the guidance policy might degrade outside those boundaries. The policy is optimized using an Actor-Critic architecture based on PPO, in which the critic network always has full access to all the information about the lander's state, whereas the actor network is provided with partial observations. Among the available observations are sequences of terrain images generated using a detailed DTM and a realistic camera model based on ray-tracing. This is achieved using a simulator built in Blender, which is interfaced with the whole RML framework through a python API. With this approach, we were able to reach a target state with position and velocity average errors within 2 m and 1 to 2 m/s, respectively, in all cases taken into consideration. Three different test cases have been analyzed: in the first one, sequences of terrain images and vertical position and velocity are provided to the actor; in the second one, sequences of terrain images, vertical position, and the full velocity vector are provided; finally, in the third case the actor has access to the full state. The results clearly show that increasing the information available to the actor increases the method's overall performance, both in terms of accuracy and precision. It is interesting to note that in cases 1 and 2 where the actor has access to reduced information about the state of the lander, the performance is surprisingly close to case 3 where the full state is fed to the actor. This proves the capabilities of recurrent networks to map sequences of partial observations to a uniquely determined state, which ultimately allows for successful guidance. Case 1 is the most interesting as it represents the most realistic case where observations could be gathered using a camera and a radar altimeter. The proposed method shows good precision in all cases, with final positions and velocities contained in tight distributions, although a slight offset is observed in the x and y components of the position and velocity. Note the the described RML approach exhibits low computational cost. Indeed, once the policy neural network is trained, the network evaluation is a simple sequence of matrix additions and



(a) Reward



(b) Test trajectories



(c) Test statistics

Fig. 12 Results for Case 3.

multiplications that can be optimized for on-board implementation. Overall, the presented method builds upon the previous work in the field, introducing optical images as observations. This approach is currently being extended to integrate autonomous hazard detection, performed via convolutional neural networks, and obstacle avoidance techniques in order to achieve a higher level of autonomy.

References

[1] Smith, M., Craig, D., Herrmann, N., Mahoney, E., Krezel, J., McIntyre, N., and Goodliff, K., "The Artemis Program: An Overview of NASA's Activities to Return Humans to the Moon," *2020 IEEE Aerospace Conference*, IEEE, 2020, pp. 1–10. https://doi.org/10.1109/AERO47225.2020.9172323.

[2] Zurbuchen, T. H., "Mars Exploration Program," *Presentation to the National Academies*, Vol. 28, 2017.

[3] Klumpp, A. R., "Apollo lunar descent guidance," *Automatica*, Vol. 10, No. 2, 1974, pp. 133–146. https://doi.org/10.1016/0005-1098(74)90019-3.

[4] Klumpp, A. R., "A manually retargeted automatic landing system for the lunar module (LM)," *Journal of Spacecraft and Rockets*, Vol. 40, No. 6, 2003, pp. 973–982. https://doi.org/10.2514/2.7044.

[5] Morrison, D. D., Riley, J. D., and Zancanaro, J. F., "Multiple shooting method for two-point boundary value problems," *Communications of the ACM*, Vol. 5, No. 12, 1962, pp. 613–614. https://doi.org/10.1145/355580.369128.

[6] Mortari, D., "The theory of connections: Connecting points," *Mathematics*, Vol. 5, No. 4, 2017, p. 57. https://doi.org/10.3390/math5040057.

[7] 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.

[8] Furfaro, R., and Mortari, D., "Least-squares solution of a class of optimal space guidance problems via Theory of Connections," *Acta Astronautica*, Vol. 168, 2020, pp. 92–103. https://doi.org/10.1016/j.actaastro.2019.05.050.

[9] Johnston, H., Schiassi, E., Furfaro, R., and Mortari, D., "Fuel-Efficient Powered Descent Guidance on Large Planetary Bodies via Theory of Functional Connections," *arXiv preprint arXiv:2001.03572*, 2020.

[10] Betts, J. T., *Practical methods for optimal control and estimation using nonlinear programming*, SIAM, 2010. https://doi.org/10.1137/1.9780898718577.ch3.

[11] Lustig, I. J., Marsten, R. E., and Shanno, D. F., "Interior point methods for linear programming: Computational state of the art," *ORSA Journal on Computing*, Vol. 6, No. 1, 1994, pp. 1–14. https://doi.org/10.1287/ijoc.6.1.1.

[12] Burke, J. V., "A robust trust region method for constrained nonlinear programming problems," *SIAM Journal on Optimization*, Vol. 2, No. 2, 1992, pp. 325–347. https://doi.org/10.1137/0802016.

[13] Singer, S., and Nelder, J., "Nelder-mead algorithm," *Scholarpedia*, Vol. 4, No. 7, 2009, p. 2928. https://doi.org/10.4249/scholarpedia.2928.

[14] Liu, X., Lu, P., and Pan, B., "Survey of convex optimization for aerospace applications," *Astrodynamics*, Vol. 1, No. 1, 2017, pp. 23–40. https://doi.org/10.1007/s42064-017-0003-8.

[15] 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.

[16] Açıkmeşe, B., Carson, J. M., and Blackmore, L., "Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem," *IEEE Transactions on Control Systems Technology*, Vol. 21, No. 6, 2013, pp. 2104–2113. https://doi.org/10.1109/tcst.2012.2237346.

[17] Blackmore, L., Açikmeşe, B., and Scharf, D. P., "Minimum-landing-error powered-descent guidance for Mars landing using convex optimization," *Journal of guidance, control, and dynamics*, Vol. 33, No. 4, 2010, pp. 1161–1171. https://doi.org/10.2514/1.47202.

[18] Pinson, R. M., and Lu, P., "Trajectory design employing convex optimization for landing on irregularly shaped asteroids," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 6, 2018, pp. 1243–1256. https://doi.org/10.2514/6.2016-5378.

[19] Johnson, A. E., Cheng, Y., Montgomery, J. F., Trawny, N., Tweddle, B., and Zheng, J. X., "Real-time terrain relative navigation test results from a relevant environment for Mars landing," *AIAA Guidance, Navigation, and Control Conference*, 2015, p. 0851. https://doi.org/10.2514/6.2015-0851.

[20] Gaskell, R. W., Barnouin-Jha, O. S., Scheeres, D. J., Konopliv, A. S., Mukai, T., Abe, S., and Kawaguchi, J., "Characterizing and navigating small bodies with imaging data," *Meteoritics & Planetary Science*, Vol. 43, No. 6, 2008, pp. 1049–1061. https://doi.org/10.1111/j.1945-5100.2008.tb00692.x.

[21] Lorenz, D. A., Olds, R., May, A., Mario, C., Perry, M. E., Palmer, E. E., and Daly, M., "Lessons learned from OSIRIS-Rex autonomous navigation using natural feature tracking," *In Aerospace Conference, 2017 IEEE*, 2017, pp. 1–12. https://doi.org/10.1109/aero.2017.7943684.

[22] Christian, J. A., Hong, L., McKee, P., Christensen, R., and Crain, T. P., "Image-Based Lunar Terrain Relative Navigation Without a Map: Measurements," *Journal of Spacecraft and Rockets*, 2020, pp. 1–18. https://doi.org/10.2514/1.a34875.

[23] NASA, *Vision for Space Exploration*, National Aeronautics and Space Administration, Washington DC, 2004. https://doi.org/10.1007/978-0-387-48758-8_10.

[24] Li, S., Jiang, X., and Tao, T., "Guidance summary and assessment of the Chang'e-3 powered descent and landing," *Journal of Spacecraft and Rockets*, Vol. 53, No. 2, 2016, pp. 258–277. https://doi.org/10.2514/1.A33208.

[25] You, S., Wan, C., Dai, R., and Rea, J. R., "Learning-Based Onboard Guidance for Fuel-Optimal Powered Descent," *Journal of Guidance, Control, and Dynamics*, Vol. 44, No. 3, 2021, pp. 601–613. https://doi.org/10.2514/1.g004928.

[26] Sánchez-Sánchez, C., and Izzo, D., "Real-time optimal control via Deep Neural Networks: study on landing problems," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, 2018, pp. 1122–1135. https://doi.org/10.2514/1.g002357.

[27] Zhu, L., Ma, J., and Wang, S., "Deep neural networks based real-time optimal control for lunar landing," *IOP Conference Series: Materials Science and Engineering*, Vol. 608, IOP Publishing, 2019, p. 012045. https://doi.org/10.1088/1757-899x/608/1/012045.

[28] Song, Y., Cheng, L., and Gong, S., "Fast estimation of gravitational field of irregular asteroids based on deep neural network and its application," *Advances in the Astronautical Sciences*, Vol. 168, 2019, pp. 1721–1735.

[29] Izzo, D., Sprague, C. I., and Tailor, D. V., "Machine learning and evolutionary techniques in interplanetary trajectory design," *Modeling and Optimization in Space Engineering*, Springer, 2019, pp. 191–210. https://doi.org/10.1007/978-3-030-10501-3_8.

[30] Furfaro, R., Bloise, I., Orlandelli, M., Di Lizia, P., Topputo, F., Linares, R., et al., "Deep learning for autonomous lunar landing," *2018 AAS/AIAA Astrodynamics Specialist Conference*, 2018, pp. 1–22.

[31] Patterson, M. A., and Rao, A. V., "GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Transactions on Mathematical Software (TOMS)*, Vol. 41, No. 1, 2014, pp. 1–37. https://doi.org/10.1145/2558904.

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

[33] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[34] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[35] Furfaro, R., and Linares, R., "Waypoint-Based Generalized ZEM/ZEV Feedback Guidance for Planetary Landing via a Reinforcement Learning Approach," *3rd IAA Conference on Dynamics and Control of Space Systems, Moscow, Russia*, 2017. https://doi.org/10.1016/j.actaastro.2020.02.051.

[36] Furfaro, R., Scorsoglio, A., Linares, R., and Massari, M., "Adaptive generalized ZEM-ZEV feedback guidance for planetary landing via a deep reinforcement learning approach," *Acta Astronautica*, 2020. https://doi.org/10.1016/j.actaastro.2020.02.051.

[37] 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.

[38] Jiang, J., Zeng, X., Guzzetti, D., and You, Y., "Path planning for asteroid hopping rovers with pre-trained deep reinforcement learning architectures," *Acta Astronautica*, Vol. 171, 2020, pp. 265–279. https://doi.org/10.1016/j.actaastro.2020.03.007.

[39] Hovell, K., and Ulrich, S., "On Deep Reinforcement Learning for Spacecraft Guidance," *AIAA Scitech 2020 Forum*, 2020, p. 1600. https://doi.org/10.2514/6.2020-1600.

[40] Oestreich, C. E., Linares, R., and Gondhalekar, R., "Autonomous Six-Degree-of-Freedom Spacecraft Docking with Rotating Targets via Reinforcement Learning," *Journal of Aerospace Information Systems*, 2021, pp. 1–12. https://doi.org/10.2514/1.i010914.

[41] Wang, X., Wang, G., Chen, Y., and Xie, Y., "Autonomous Rendezvous Guidance via Deep Reinforcement Learning," *2020 Chinese Control And Decision Conference (CCDC)*, IEEE, 2020, pp. 1848–1853. https://doi.org/10.1109/ccdc49329.2020.9163988.

[42] Holt, H., Armellin, R., Scorsoglio, A., and Furfaro, R., "Low-thrust Trajectory Design using Closed-loop Feedback-driven Control Laws and State-dependent Parameters," *AIAA Scitech 2020 Forum*, 2020, p. 1694. https://doi.org/10.2514/6.2020-1694.

[43] Zavoli, A., and Federici, L., "Reinforcement Learning for Low-Thrust Trajectory Design of Interplanetary Missions," *arXiv preprint arXiv:2008.08501*, 2020.

[44] Arora, L., and Dutta, A., "Reinforcement Learning for Sequential Low-Thrust Orbit Raising Problem," *AIAA Scitech 2020 Forum*, 2020, p. 2186. https://doi.org/10.2514/6.2020-2186.

[45] LaFarge, N. B., Miller, D., Howell, K. C., and Linares, R., "Guidance for closed-loop transfers using reinforcement learning with application to libration point orbits," *AIAA Scitech 2020 Forum*, 2020, p. 0458. https://doi.org/10.2514/6.2020-0458.

[46] Silvestrini, S., and Lavagna, M. R., "Spacecraft Formation Relative Trajectories Identification for Collision-Free Maneuvers using Neural-Reconstructed Dynamics," *AIAA Scitech 2020 Forum*, 2020, p. 1918. https://doi.org/10.2514/6.2020-1918.

[47] Scorsoglio, A., Furfaro, R., Linares, R., Massari, M., et al., "Actor-critic reinforcement learning approach to relative motion guidance in near-rectilinear orbit," *29th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Soc. San Diego, CA, 2019, pp. 1–20.

[48] Scorsoglio, A., and Furfaro, R., "ELM-based Actor-Critic Approach to Lyapunov Vector Fields Relative Motion Guidance in Near-Rectilinear Orbit," *2019 AAS/AIAA Astrodynamics Specialists Conference*, 2019, pp. 1–20.

[49] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. URL https://www.arXiv:1509.02971.

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

[51] Finn, C., Abbeel, P., and Levine, S., "Model-agnostic meta-learning for fast adaptation of deep networks," *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR.org, 2017, pp. 1126–1135.

[52] 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*, 2020. https://doi.org/10.1016/j.actaastro.2020.03.026.

[53] 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.

[54] Gaudet, B., Linares, R., and Furfaro, R., "Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations," *Acta Astronautica*, 2020. https://doi.org/10.1016/j.actaastro.2020.02.036.

[55] Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M., "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

[56] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y., "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014. URL https://www.arXiv:1412.3555.

[57] Williams, R. J., "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, Vol. 8, No. 3-4, 1992, pp. 229–256. https://doi.org/10.1007/978-1-4615-3618-5_2.

[58] Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al., "Policy gradient methods for reinforcement learning with function approximation." *NIPs*, Vol. 99, Citeseer, 1999, pp. 1057–1063.

[59] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P., "Trust region policy optimization," *International conference on machine learning*, 2015, pp. 1889–1897.

[60] Kullback, S., and Leibler, R. A., "On information and sufficiency," *The annals of mathematical statistics*, Vol. 22, No. 1, 1951, pp. 79–86. URL https://www.jstor.org/stable/2236703.

[61] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P., "OpenAI Baselines," https://github.com/openai/baselines, 2017.

[62] Prazdny, K., "Egomotion and relative depth map from optical flow," *Biological cybernetics*, Vol. 36, No. 2, 1980, pp. 87–102. https://doi.org/10.1007/bf00361077.

[63] Tchernykh, V., Beck, M., Janschek, K., and Flandin, G., "An Optical Flow Approach for Precise Visual Navigation of a Planetary Lander," *Guidance, Navigation and Control Systems*, Vol. 606, 2006.

[64] Hare, J., "Dealing with sparse rewards in reinforcement learning," *arXiv preprint arXiv:1910.09281*, 2019.