IAC–24–C1,5,7

Autonomous Image-based Navigation in Cislunar Orbits via Meta-Reinforcement Learning

Elia Violino[a*], Andrea Scorsoglio[b], Luca Ghilardi[c], Lorenzo Federici[d], Francesco Topputo[e], Roberto Furfaro[f]

[a] *Department of Aerospace Science and Technology, Politecnico di Milano, Via La Masa 34, 2016 Milano, Italy, elia.violino@mail.polimi.it*
[b] *Department of System and Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721, andreascorsoglio@email.arizona.edu*
[c] *Department of System and Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721, lucaghilardi@email.arizona.edu*
[d] *Department of System and Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721, lorenzof@email.arizona.edu*
[e] *Department of Aerospace Science and Technology, Politecnico di Milano, Via La Masa 34, 2016 Milano, Italy, francesco.topputo@polimi.it*
[f] *Department of System and Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721, robertof@email.arizona.edu*
[*] *Corresponding author*

Abstract

Spacecraft navigation is a vital component of any space mission. Image-based navigation is a promising solution for autonomous real-time navigation applications in deep space exploration missions. It typically involves computer vision techniques and mathematical models to determine the spacecraft's position, velocity, and orientation based on images taken by the onboard cameras. Traditional algorithms for image-based navigation include feature-based navigation and structure-from-motion. However, these methods typically require high onboard computational power and the availability of high-resolution images, which can be challenging to obtain in certain low-light or limited-visibility conditions. In this paper, reinforcement learning (RL) is proposed as a novel methodology for autonomous image-based spacecraft navigation by employing a convolutional neural network as a trajectory estimation policy. In the context of image-based navigation, the convolutional network and a multi-layer perceptron can process image data from a simulated lunar environment developed in computer graphic software and adjust their estimates of the current spacecraft's position accordingly. With RL the policy is trained via repeated interaction with a simulated environment, progressively learning to adapt to changes in the environment, to deal with noise in the collected images and inaccuracies in the dynamical model. This framework is applied to the autonomous navigation of a spacecraft along a L_2 south Halo orbit in cislunar space. The method is able to provide a good estimation of the spacecraft's position, maintaining the error within admissible values for an entire orbit, thus proving that RL is a powerful and effective method for image-based autonomous navigation.

Nomenclature

α	learning rate
β	position angle of EMS
$\boldsymbol{\pi}_\theta$	policy
$\boldsymbol{a}$	policy's action vector
$\boldsymbol{C}_{\mathcal{I}}^{\mathcal{S}}$	non-dimensional transformation matrix from $\mathcal{I}$ to $\mathcal{S}$
$\boldsymbol{i}$	image
$\boldsymbol{r}$	position vector
$\boldsymbol{r}_\oplus$	position vector of the EMS with respect to the Sun
$\boldsymbol{T}$	thrust vector
$\boldsymbol{x}$	state vector
$\boldsymbol{y}$	observation vector
δ_{stop}	position threshold
γ	discount factor
$\hat{}$	estimate
$\mathcal{A}$	action space
$\mathcal{B}$	Blender reference frame
$\mathcal{I}$	inertial reference frame centered on the EMB
$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$	Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance Σ
$\mathcal{S}$	synodic reference frame
$\mathcal{U}(a, b)$	uniform distribution in interval $[a, b]$
$\mathcal{X}$	state space
$\mathcal{Y}(\boldsymbol{x}_h)$	observation function
$\mathbb{E}_\tau[R]$	expected value of $R(\tau)$ with respect

	to random variable τ
max	maximum value
μ	mass ratio parameter
ν	true anomaly of the Moon's orbit
σ_x	standard deviation of Gaussian random variable x
τ	trajectory
θ	neural network's parameters
$\top$	transpose operator
$\star$	optimal value
A^π	advantage function
a_{pert}	magnitude of the perturbation
H	number of steps per episode
h	value at h-th step
I_{sp}	specific impulse
$J(\theta)$	objective function
L	characteristic length of the EMS
$L(w)$	loss function for the critic
m	spacecraft's mass
$R(\boldsymbol{x}_h, \boldsymbol{a}_h)$	reward function
r_h	reward at step h
t	time
U	non-dimensional potential
w_{adj}	reward weight related to the action
w_{pen}	reward weight related to the penalty due to early truncation
w_{pos}	reward weight related to the position
$\boldsymbol{R}_a^b$	rotation matrix from reference frame b to a
$P(\boldsymbol{x}_{h+1} \mid \boldsymbol{x}_h, \boldsymbol{a}_h)$	state transition distribution

1. Introduction

Space navigation is the process of determining the state of a spacecraft in space. It is a paramount component of every space mission and employs a combination of onboard sensors and ground-based observations. Nowadays, typical current deep space navigation approaches are based on ground-based tracking that provides radiometric observables to estimate the spacecraft's state. Furthermore, maneuvering commands are given from a ground station, leading to completely ground-dependent vehicles with possibly high delays between command and actuation [1, 2]. From the ground segment's perspective, small satellite operations pose several challenges, including the need to track multiple spacecraft simultaneously within limited contact windows, managing a growing number of missions with finite tracking resources, addressing power constraints, and dealing with the associated operational costs, including those of flight dynamics teams.

The growing interest in deep space exploration has increased the need for more autonomous missions, with reliance on the ground segment becoming an additional obstacle. Autonomous navigation presents a promising solution, enabling spacecraft to operate independently and make decisions without human intervention. Furthermore, the capability of autonomous navigation is essential for addressing events that occur on time scales shorter than the communication latency between Earth and the spacecraft [3].

Optical navigation has gained significant attention in recent years to attain reliable and efficient space navigation. This method involves using on-board camera sensors to estimate the relative position and velocity between the spacecraft and target bodies. By leveraging knowledge about the surrounding environment, optical navigation has been employed to complement traditional navigation techniques, particularly during missions near celestial bodies [4, 5].

One category of optical navigation methods includes triangulation schemes, that is, determining the observer location as the intersection point between line-of-sight directions to target locations. These methods are primarily employed in relative line-of-sight navigation for detecting and approaching celestial bodies [6–8], in relative navigation between spacecraft [9], and in surface navigation using target features [10]. Another class of optical navigation methods relies on prospective projection, which exploits the relation between the apparent size of an observed object in the camera's field of view and its relative distance to it [11]. By knowing the true size of the object, the relative distance can be estimated by comparing the actual size to the apparent one. Additionally, horizon-based navigation provides valuable information by using a body's horizon to estimate the distance to the object. The relative position vector is then determined by identifying the object's location within the camera's field of view [12, 13].

While optical autonomous navigation has significantly enhanced spacecraft state estimation, the integration with Artificial Intelligence (AI) introduced a transformative element to space exploration, greatly enhancing autonomous capabilities. Specifically, the fusion of optical navigation with AI opened up new possibilities for spacecraft to process vast datasets, enabling intelligent and rapid decision-making and boosting adaptability across various environments. Neural Networks (NNs) offer the potential to drastically cut down computational time by being trained on the ground beforehand, thereby enabling near-instantaneous in-flight responses. Furthermore, NN generalization capabilities allow for adjustments to changing environmental conditions, leveraging the ability to learn from past experiences.

In space navigation, Machine Learning (ML) methods are predominantly applied in image processing via

Convolutional Neural Networks (CNNs), while the navigation tasks themselves are typically handled by the well-established Kalman Filter (KF) [14]. CNNs are also effectively used in space exploration for pose estimation of non-cooperative targets, yielding valuable and effective results [15–17]. There have also been efforts to replace the Kalman Filter with a Neural Network [18–20] and to integrate NN with KF [21], though these approaches have not been specifically tailored for space applications. A notable example of space navigation using neural networks can be found in Ref. [22], where a Convolutional Extreme Learning Machine (CELM) is utilized for vision-based navigation to assign a range label to each input image.

One significant advancement in AI is Reinforcement Learning (RL), a machine learning technique that enables an agent to learn an optimal control policy by maximizing the cumulative reward received for its actions within an environment. RL has diverse applications, including robotics, image processing, and control, and its versatility has made it widely adopted also in space engineering. In particular, RL has been used to address spacecraft guidance and control problems, such as autonomous guidance of an asteroid impactor [23], spacecraft guidance during rendezvous missions [24], autonomous lunar landing [25, 26], or relative orbit spacecraft docking [27], obtaining promising results.

In this paper, the objective is to apply Reinforcement Learning (RL) to develop a neural network that maps, in a closed-loop manner, images of the space environment to corrections in the spacecraft's estimated position, ultimately refining the knowledge available onboard about its true position. The initial position estimate is obtained by propagating the spacecraft's state from the previous time step. This presents a unique challenge for RL, which is traditionally designed for decision-making in dynamic systems where control policies generate actions that directly affect the environment and, consequently, the system's state. In this context, however, RL generates an action that doesn't directly modify the system state but instead provides a correction to an estimated position, which serves as input for the control system onboard. This indirect approach adds an additional layer of complexity, as the policy must adjust a proxy value rather than directly interacting with the true system state. Precisely, we apply this RL-based framework to perform optical navigation in an L_2 Southern Halo orbit around the Moon. To achieve this, synthetic images of the Moon were generated using Blender, which uses a physically-based rendering engine to simulate realistic images as those captured by an onboard optical camera. These images are used as input to train a neural network, consisting of both convolutional and fully-connected layers, which is responsible for correcting the estimated spacecraft's position. Key parameters about the environment, such as solar position and initial orbital conditions of the spacecraft, are sampled at the beginning of each simulation from stochastic distributions, allowing the agent to generalize across diverse environmental conditions. The result is a navigation system capable of providing accurate position estimates within an image-based feedback loop.

The structure of the paper is as follows: Section 2 outlines the machine learning framework, particularly RL, and its implementation in our approach. Section 3 details the implementation of the RL-based navigation algorithm, presenting the dynamic model, the MDP formulation, and the visual environment. Section 4 discusses the performance of the proposed algorithm across various configurations. Lastly, Section 5 provides concluding remarks and summarizes the key findings.

2. Machine Learning Framework

In the following section, the RL formulation is introduced as well as the foundation for the learning procedure based on the PPO algorithm [28] used in this work.

2.1 Reinforcement Learning

In reinforcement learning (RL), an agent learns to perform tasks by repeatedly interacting with an environment. This interaction framework is typically modeled as a Markov Decision Process (MDP), where the state depends solely on the preceding state and the applied action at any given time. The MDP serves as a mathematical representation of the environment and is defined by a continuous state space $\mathcal{X}$, an action space $\mathcal{A}$, a state transition distribution $P(\boldsymbol{x}_{h+1} \mid \boldsymbol{x}_h, \boldsymbol{a}_h)$, which defines the probability of transitioning to the next state $\boldsymbol{x}_{h+1}$ given a specific action $\boldsymbol{a}_h$ applied to state $\boldsymbol{x}_h$, and a scalar reward function $r_h = R(\boldsymbol{x}_h, \boldsymbol{a}_h)$, where $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{a} \in \mathcal{A}$. When the state is not directly observable or the observations are noisy, the problem is modeled as a Partially Observable Markov Decision Process (POMDP). In a POMDP, the state $\boldsymbol{x}$ is hidden, and observations $\boldsymbol{y}$ are provided via an observation function $\mathcal{Y}(\boldsymbol{x})$. The agent uses a parameterized policy $\boldsymbol{\pi}_\theta$ to operate within the environment defined by the POMDP, generating an action $\boldsymbol{a}_h$ based on the observation $\boldsymbol{y}_h$, receiving a reward r_h, and transitioning to the next observation $\boldsymbol{y}_{h+1}$. The objective of the RL algorithm is to optimize the policy $\boldsymbol{\pi}_\theta$ to maximize the cumulative rewards collected over an episode [25]

$$\theta^\star = \underset{\theta}{\operatorname{argmax}} \underset{\boldsymbol{\pi}_\theta(\tau)}{\mathbb{E}} [R(\tau)] \qquad [1]$$

with $\theta^\star$ being the optimal parameters of the policy $\boldsymbol{\pi}_\theta$ and $R(\tau)$ the cumulative reward over an episode, or trajectory,

$\tau = \{(\boldsymbol{y}_0, \boldsymbol{a}_0), \ldots, (\boldsymbol{y}_{H-1}, \boldsymbol{a}_{H-1}), \boldsymbol{x}_H\}$, that is

$$R(\tau) = \sum_{h=0}^{H-1} \gamma^h r_h \quad [2]$$

with $\gamma \in [0, 1)$ a discount factor for future rewards, r_h the reward at step h, and H the total number of steps in a single episode.

At the beginning of the training process, the agent has no information about the environment and the policy is randomly initialized. To allow the agent to explore the state and action spaces and gather information about the environment, exploration is allowed. To this aim, a diagonal multivariate Gaussian policy is employed to strike a balance between exploration and exploitation during training. At each time step h, the network $\boldsymbol{\pi}_\theta$ receives the current observation $\boldsymbol{y}_h$ as input and outputs the mean action value $\boldsymbol{\mu}_h$ and the corresponding standard deviation $\boldsymbol{\sigma}_h$. Since the policy is stochastic, the notation $\boldsymbol{\pi}_\theta(\cdot \mid \boldsymbol{y}_h)$ is typically used in modern RL literature to represent the probability of selecting a specific action $\boldsymbol{a}_h$, given the observation $\boldsymbol{y}_h$. For consistency with RL terminology, this notation is also adopted in this paper, even though the policy $\boldsymbol{\pi}_\theta$ technically returns the parameters of the probability distribution rather than the probability value itself. To enable broad exploration of the solution space during training, the actual action is sampled according to the Gaussian distribution

$$\boldsymbol{a}_h \sim \boldsymbol{\pi}_\theta(\cdot \mid \boldsymbol{y}_h) \sim \mathcal{N}(\boldsymbol{\mu}_h, \Sigma_h) \quad [3]$$

with $\Sigma_h = \text{diag}(\boldsymbol{\sigma}_h \boldsymbol{\sigma}_h^\top)$ being the covariance matrix. To ensure that the action $\boldsymbol{a}_h$ always lies within its defined interval, the probability that any of its components falls outside the action space $\mathcal{A}$ (i.e., the tails of the Gaussian) is clipped to zero.

By deviating from the nominal policy, the agent can explore various possibilities, gather valuable information, and progressively refine its policy based on the outcomes of these explorations. This process is essential for the agent to discover and learn effective strategies within the environment. As the agent learns, it identifies which actions are most favorable for a given observation based on the reward, which reflects the effectiveness of each action. Once training is complete, during the final policy deployment or evaluation, exploration is disabled, and the agent returns the optimal action for each observation: $\boldsymbol{a}_h = \boldsymbol{\mu}_h$.

2.2 Proximal Policy Optimization

In this paper, we used a derivation of the A2C method to optimize the policy, the Proximal Policy Optimization (PPO) [28]. This is a popular algorithm in the family of policy gradient methods and has achieved state-of-the-art performance across various benchmark reinforcement learning tasks. PPO is derived from the Trust Region Policy Optimization (TRPO) method [29]. The TRPO algorithm formulates the policy optimization task in a way that constrains the size of the gradient step taken in each iteration using a dynamically computed constraint. The TRPO policy update can be formulated as:

$$\begin{aligned} &\min_\theta \mathop{\mathbb{E}}_{p(\tau)} \left[\frac{\pi_\theta(\boldsymbol{a}_h \mid \boldsymbol{x}_h)}{\pi_{\theta_{\text{old}}}(\boldsymbol{a}_h \mid \boldsymbol{x}_h)} A_\phi^\pi(\boldsymbol{x}_h, \boldsymbol{a}_h) \right] \\ &\text{s.t.} \mathop{\mathbb{E}}_{p(\tau)} \left[K_L \left(\pi_\theta(\boldsymbol{a}_h \mid \boldsymbol{x}_h), \pi_{\theta_{\text{old}}}(\boldsymbol{a}_h \mid \boldsymbol{x}_h) \right) \right] \leq \delta \end{aligned} \quad [4]$$

where K_L represents the Kullback-Leibler divergence [30] between the current policy and the previous policy, and δ is a parameter that constrains the update step size. It has been proven that if the update at each iteration is bounded by a constant $C(K_L)$, the policy will improve monotonically toward the optimal policy. However, such an approach typically leads to very small updates, so the formulation in Eq. 4 with a fixed constraint parameter is used instead. This optimization problem is approximately solved using the conjugate gradient method, which linearizes the objective function and applies a quadratic approximation to the constraint.

The PPO method simplifies the TRPO optimization by incorporating the constraint on policy updates directly into the objective function through clipping. The objective can be expressed in terms of the probability ratio $p_h(\theta)$ defined as:

$$p_h(\theta) = \frac{\pi_\theta(\boldsymbol{a}_h \mid \boldsymbol{x}_h)}{\pi_{\theta_{\text{old}}}(\boldsymbol{a}_h \mid \boldsymbol{x}_h)} \quad [5]$$

The clipped objective function is then defined as

$$J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\min \left(p_h \hat{A}_h, \text{clip}(p_h, 1-\epsilon, 1+\epsilon) \hat{A}_h \right) \right] \quad [6]$$

The advantage function cannot be computed exactly, thus its value is approximated as the difference between the empirical return (i.e., the discounted reward) and a baseline provided by the state value function. This advantage function indicates how much better a particular action is compared to the average action:

$$\hat{A}_\phi^\pi(\boldsymbol{x}_h, \boldsymbol{a}_h) = \left[\sum_{l=h}^{H-1} \gamma^{l-h} r_l \right] - \hat{V}_\phi^\pi(\boldsymbol{x}_h) \quad [7]$$

with $\gamma \in [0, 1)$ being the discount factor, with values closer to one indicating that the algorithm places more importance on rewards received further into the future. The subscript ϕ emphasizes that the advantage function is dependent on the value function approximation provided by the critic. The value function $\hat{V}_\phi^\pi(\boldsymbol{x}_h)$ is learned using the

following cost function:

$$L(\phi) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\hat{V}_\phi^\pi(\boldsymbol{x}_{h+1}) - \left(\sum_{l=h}^{H}\gamma^{l-h}r_l\right)\right] \quad [8]$$

In practice, policy gradient algorithms refine the policy by updating it using a batch of trajectories (also known as roll-outs) obtained from interactions with the environment. Each trajectory corresponds to a single episode, where a sample from a trajectory at step h includes the observation $\boldsymbol{x}_h$, the action $\boldsymbol{a}_h$, and the reward $r_h(\boldsymbol{x}_h, \boldsymbol{a}_h)$. The learning process involves performing gradient ascent on the policy parameters $\boldsymbol{\theta}$ and gradient descent on the value function parameters $\boldsymbol{\phi}$. The updated equations are as follows:

$$\boldsymbol{\phi}^+ = \boldsymbol{\phi}^- - \alpha_\phi \nabla_\phi L(\phi)\big|_{\boldsymbol{\phi}=\boldsymbol{\phi}^-} \quad [9]$$

$$\boldsymbol{\theta}^+ = \boldsymbol{\theta}^- + \alpha_\theta \nabla_\theta J(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^-} \quad [10]$$

with α_ϕ and α_θ representing the learning rates for the value function V_ϕ^π and the policy $\pi_\theta(\boldsymbol{a}_h \mid \boldsymbol{x}_h)$, respectively. Both the policy and the value function are updated concurrently.

It's also important to note that the parameters $\boldsymbol{\theta}$ depend on the architecture of the neural network used. In this work, we employ a policy described by a Convolutional Neural Network (CNN) followed by a Multilayer Perceptron (MLP). The training process is summarized in the pseudocode provided in Algorithm 1.

Algorithm 1 Proximal Policy Optimization (PPO).

```
initialize policies π_θ, π_θold
for iteration= 1, 2, ... do
    for episode= 1, 2, ... do
        run current policy π_θ in environment and collect trajectory τ_i
    end for
    calculate advantage function using Eq. 7
    update the value function parameters using Eq. 9
    calculate clipped objective using Eq. 6
    update the policy parameters using Eq. 10
end for
```

3. Reinforcement Learning Navigation Algorithm Implementation

3.1 Dynamic model

The spacecraft dynamic comes from the Circular Restricted Three-Body Problem (CR3BP), where the Earth is the primary body and the Moon is the secondary one. For simplicity, the Moon is assumed to orbit on the Earth's equatorial plane.

The equations governing the dynamics of the CR3BP are formulated in the non-dimensional synodic reference frame. The behavior of the system is primarily influenced by the mass ratio parameter μ, defined as:

$$\mu = \frac{m_2}{m_1 + m_2} \quad [11]$$

Within this reference frame, the equations of motion for the third body, the spacecraft, are given by:

$$\begin{cases} \ddot{x} - 2\dot{y} = x - \dfrac{1-\mu}{{r_1}^3}(x+\mu) - \dfrac{\mu}{{r_2}^3}(x-(1-\mu)) \\ \ddot{y} + 2\dot{x} = y - y\left(\dfrac{1-\mu}{{r_1}^3} + \dfrac{\mu}{{r_2}^3}\right) \\ \ddot{z} = -z\left(\dfrac{1-\mu}{{r_1}^3} + \dfrac{\mu}{{r_2}^3}\right) \end{cases} \quad [12]$$

where the distances r_1 and r_2 are defined as:

$$r_1 = \sqrt{(x+\mu)^2 + y^2 + z^2} \quad [13]$$

$$r_2 = \sqrt{(x-(1-\mu))^2 + y^2 + z^2} \quad [14]$$

By introducing the non-dimensional potential U, defined as:

$$U = \frac{1}{2}(x^2 + y^2) + \frac{1-\mu}{r_1} + \frac{\mu}{r_2} \quad [15]$$

the equations of motion can be rewritten as:

$$\begin{cases} \ddot{x} - 2\dot{y} = \dfrac{\partial U}{\partial x} \\ \ddot{y} + 2\dot{x} = \dfrac{\partial U}{\partial y} \\ \ddot{z} = \dfrac{\partial U}{\partial z} \end{cases} \quad [16]$$

To maintain the spacecraft close to the target Halo orbit, a control force is required due to the unstable nature of these orbits. The control thrust vector is defined as $\boldsymbol{T} = [T_x, T_y, T_z]^\top$, having a maximum magnitude $T_{\max}$. The effective exhaust velocity is given by $u_{eq} = g_0 I_{sp}$, with $g_0 = 9.807$ m/s^2 representing the Earth's gravitational acceleration at sea level. The equations of motion with control can be expressed as:

$$\begin{cases} \ddot{x} - 2\dot{y} = \dfrac{\partial U}{\partial x} + \dfrac{T_x}{m}, \\ \ddot{y} + 2\dot{x} = \dfrac{\partial U}{\partial y} + \dfrac{T_y}{m}, \\ \ddot{z} = \dfrac{\partial U}{\partial z} + \dfrac{T_z}{m}, \\ \dot{m} = -\dfrac{||\boldsymbol{T}||}{u_{eq}}, \end{cases} \quad [17]$$

where m denotes the mass of the spacecraft.

3.2 Markov Decision Process

The problem of image-based navigation is formulated as a Markov Decision Process (MDP). Time is discretized into a grid of H uniformly spaced time steps, starting from the initial time t_0 to a maximum time $t_{\max} = t_H$, with each time step denoted by t_h.

The spacecraft's state at each time step t_h is obtained by numerically integrating the equation of motion (Eq. [17]) from t_{h-1} to t_h. During this interval, the thrust is assumed to be constant in both magnitude and direction.

A Halo orbit is employed as a reference trajectory for this navigation problem. To keep the spacecraft near the desired orbit, a closed-loop control law is used, which achieves an accuracy of 30 km in position and 0.3 m/s in velocity when trained on the specific Halo orbit [31].

The control law, however, requires accurate state estimates of the spacecraft. These estimates are provided by the policy $\boldsymbol{\pi}$ for position and a simulated generic navigation subsystem for velocity. At each time step, the navigation system takes a vector of observations $\boldsymbol{y}_h$ as input, represented by:

$$\boldsymbol{y}_h = \begin{bmatrix} \boldsymbol{i}_h \\ \beta_h \end{bmatrix}, \tag{18}$$

where $\boldsymbol{i}_h$ is a 128×128 image and β_h is the sun's position angle. The system outputs a correction for the estimated position:

$$\boldsymbol{a}_h = \boldsymbol{\pi}(\boldsymbol{y}_h) = \Delta \boldsymbol{r}_h. \tag{19}$$

The corrected position is then computed as:

$$\hat{\boldsymbol{r}}_h^+ = \hat{\boldsymbol{r}}_h^- + \boldsymbol{a}_h, \tag{20}$$

where $\hat{\boldsymbol{r}}_h^-$ is the propagated position estimate from the previous time step. Velocity estimation is obtained by adding random noise, with a standard deviation of 0.3 m/s, to each component of the real velocity, simulating the effect of a navigation subsystem dedicated to velocity estimation.

The new state estimate at time step $h+1$ is updated by performing a numerical integration of Eq. 17, yielding $\hat{\boldsymbol{x}}_{h+1}^-$. Figure 1 illustrates the proposed predictor-corrector steps, where $\boldsymbol{r}_h$ represents the spacecraft's true position.

At the start of each episode, the initial condition along the Halo orbit is randomly selected from a uniform distribution over the H possible starting points:

$$l = \mathcal{U}(\{1, \ldots, H\}). \tag{21}$$

The spacecraft's initial state is determined by adding small random errors in position and velocity to the nominal initial condition along the Halo orbit $\bar{\boldsymbol{x}}_l$:

$$\boldsymbol{x}_0 \sim \mathcal{U}\left(\bar{\boldsymbol{x}}_l - \delta \boldsymbol{x}_{0,\max},\, \bar{\boldsymbol{x}}_l + \delta \boldsymbol{x}_{0,\max}\right). \tag{22}$$

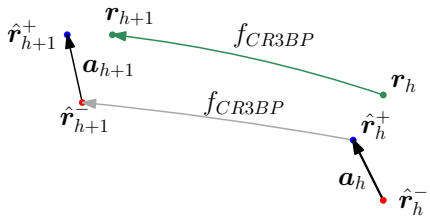

Fig. 1. Proposed predictor-corrector method.

An episode ends when one of the following two conditions is met:

$$\begin{aligned} \xi_{dist}(\hat{\boldsymbol{r}}_h^+, \boldsymbol{r}_h) &= [||\hat{\boldsymbol{r}}_h^+ - \boldsymbol{r}_h|| \geq \delta_{stop}] \\ \xi_{time}(t_h) &= [t_h \geq t_H], \end{aligned} \tag{23}$$

where the first condition terminates the episode if the distance between the estimated and actual positions exceeds the threshold δ_{stop}, and the second condition ends the episode when t_h reaches the final time $t_H = t_{\max}$, corresponding to the period of the Halo orbit.

At the end of each time step h, the agent receives a reward R_h that evaluates the quality of the position correction $\hat{\boldsymbol{r}}_h^-$ based on the observation $\boldsymbol{y}_h$. The reward function is defined as follows:

$$R_h = \begin{cases} -\delta_{err} & \text{if } \delta_{err} > 0, \\ w_{adj} w_{pos} ||\boldsymbol{a}_h|| & \text{if } \delta_{err} = 0, \\ -w_{pen} w_{pos} \delta_{stop} (H - h) & \text{if } \xi_{dist} \text{ is true}, \end{cases} \tag{24}$$

where

$$\delta_{err} = w_{pos} \max(||\hat{\boldsymbol{r}}_h^+ - \boldsymbol{r}_h|| - \epsilon_r, 0). \tag{25}$$

If the navigation error exceeds the target error ϵ_r, the policy receives a negative reward proportional to the error. If the error is within the target, the reward is positive and increases with the magnitude of the correction $\boldsymbol{a}_h$. If the episode is terminated due to the condition ξ_{dist}, the reward corresponds to the maximum distance δ_{stop} for each remaining time step. The negative reward for exceeding the navigation error is intended to encourage the policy to minimize the error. The penalty for early termination discourages highly inaccurate policies, while the positive reward encourages larger corrections, preventing the agent from settling into local minima with small, uncorrected errors.

3.3 Network Architecture

In this section, the network architecture is presented. As said before, this work employs separate actor and critic networks. Both networks consist of a deep neural network

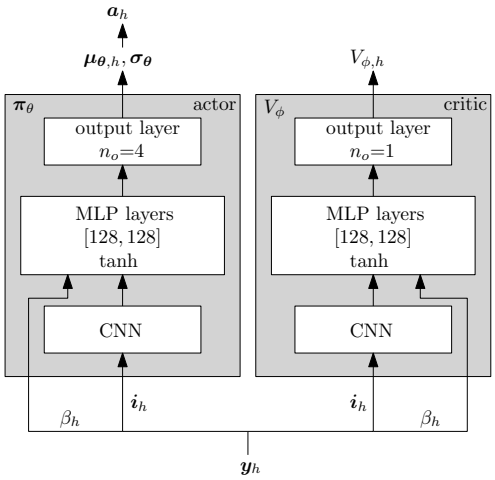

Fig. 2. Neural Network architecture.

Fig. 3. Render of the Moon.

that includes a convolutional neural network (CNN) and two fully connected (MLP) hidden layers (see Fig. 2). The CNN is structured with three convolutional filters and processes a 128×128 tensor $\boldsymbol{i}_h$ as input, which contains the image data. The output of the CNN is then fed into the MLP layers, which have 128 neurons each. These MLP layers take as input the processed image data from the CNN and the value of the sun's position angle, $\beta_\odot$. This results in an overall input size of 129 values for the MLP layers.

In the policy network, the output from the MLP layers is directed into an output layer that returns the mean value $\boldsymbol{\mu}_{\boldsymbol{\theta},h}$ for four output variables: the magnitude of the adjustment on the position and the three components of the adjustment unit vector. The standard deviation $\boldsymbol{\sigma}_{\boldsymbol{\theta}}$ is not dependent on the specific observation, making the total number of outputs for the actor network $n_o = 4$.

In contrast, the critic network's output is a single value, representing the value function $V_{\phi,h}$. This value function evaluates the expected return from a given state, helping to assess the quality of the policy generated by the actor network.

3.4 Visual Environment

The spacecraft is assumed to have an optical sensor (i.e. a camera) that provides the policy with images taken onboard. The optical sensor's foresight is always pointed to the center of the Moon. The images coming from the camera are generated within a simulated environment built in the open-source software Blender. Specifically, to recreate the surface features, the Moon is modeled as a sphere where the surface consists of a precise texture wrapped on the sphere and a bump map to simulate the shadows.

To represent the whole navigation problem inside the Blender environment, a new reference frame was introduced. Indeed, keeping the synodic reference frame would have led to the mean radius of the Moon being too small, and considering that the ray tracing-based render works better with larger numbers because of the roundoff error, this would have led to a suboptimal choice. For this reason, some modifications have been applied to the synodic frame. First of all, the unit has been set to thousands of kilometers ($\ell = 1000$ km), which allows the creation of the moon with a radius of 1.738 in the rendering space. Then, the origin of the Blender reference system is positioned at the center of the Moon, allowing for the repositioning of both the satellite and the Sun relative to this fixed point. In this setup, the Moon remains stationary, and the Earth is never within the satellite's view along its orbit. This approach has been preferred and implemented in the model. Consequently, all position vectors must be transformed from the synodic reference frame $\mathcal{S}$ to the new Blender reference frame $\mathcal{B}$.

The conversion of the position vector to the Blender reference frame $\mathcal{B}$ is carried out as follows:

$$\boldsymbol{r}^{\mathcal{B}} = \left(\boldsymbol{r}^{\mathcal{S}} - \boldsymbol{r}_{moon}{}^{\mathcal{S}}\right) \frac{L}{\ell} \qquad [26]$$

with $L = 384400$ km being the characteristic length of the Earth-Moon system. Here, the position vector $\boldsymbol{r}^{\mathcal{S}}$ in the synodic frame is adjusted by subtracting the Moon's position vector $\boldsymbol{r}_{moon}{}^{\mathcal{S}}$ in the same frame, and then scaled by the ratio $\frac{L}{\ell}$ to obtain the position vector $\boldsymbol{r}^{\mathcal{B}}$ in the Blender reference frame.

The direction of the Sun in the visual environment is used to establish the direction in which the light acts on the Moon. The direction of the sun is calculated considering that the EMS revolves in a circular orbit around it. Firstly, to determine the position of the Earth-Moon system ($\boldsymbol{r}_\oplus$) relative to the Sun, it is essential to compute its parameters in a circular orbit, specifically the radius r and the angle β

(see Figure 4). Next, a reference epoch must be established

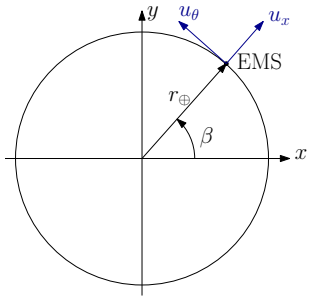

Fig. 4. EMS circular orbit

to synchronize the dynamics of the Halo orbit with the Earth-Moon system's revolution around the Sun. In this setup, β is set to β_0 at the initialization of each episode.

As time t progresses, the angle β is computed as:

$$\beta = \beta_0 + \frac{2\pi t}{T_{\text{earth-sun}}} \quad [27]$$

where t represents the elapsed time, β_0 is the initial angle, and $T_{\text{earth-sun}}$ is the period of the Earth-Moon barycenter (EMB) orbit around the Sun, approximately 365.256 days.

To derive the position vector of the Sun in the inertial reference frame $\mathcal{I}$ centered on the EMB, we perform a sequence of two transformations using the following rotation matrices:

$$\boldsymbol{R}_{\text{rot}}^{\text{ecl}} = \begin{bmatrix} \cos(\beta) & \sin(\beta) & 0 \\ -\sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [28]$$

$$\boldsymbol{R}_{\text{ecl}}^{\mathcal{I}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(i) & \sin(i) \\ 0 & -\sin(i) & \cos(i) \end{bmatrix} \quad [29]$$

where $\boldsymbol{R}_{\text{rot}}^{\text{ecl}}$ represents the rotation from the ecliptic reference frame to the rotating frame, and $\boldsymbol{R}_{\text{ecl}}^{\mathcal{I}}$ is the rotation matrix from the inertial frame to the ecliptic plane, with $i = 23.439°$ being the inclination between these planes.

The position vector of the Sun in the inertial reference frame is then given by:

$$\boldsymbol{r}_{\odot}^{\mathcal{I}} = \left(\boldsymbol{R}_{\text{rot}}^{\text{ecl}} \boldsymbol{R}_{\text{ecl}}^{\mathcal{I}}\right)^{\top} \boldsymbol{r}_{\odot}^{(\text{rot})} \quad [30]$$

where $\boldsymbol{r}_{\odot}^{(\text{rot})} = \begin{bmatrix} -r_\oplus & 0 & 0 \end{bmatrix}$ is the position vector of the Sun relative to the EMB in the rotating frame (Figure 4), with $r_\oplus = 149.60 \times 10^6$ km.

After obtaining the position in the inertial frame $\mathcal{I}$, it must be transformed into the synodic reference frame. This is achieved using the non-dimensional transformation matrix from the inertial frame $\mathcal{I}$ to the synodic frame $\mathcal{S}$:

$$\boldsymbol{\rho}^{\mathcal{S}} = \boldsymbol{C}_{\mathcal{I}}^{\mathcal{S}} \boldsymbol{\rho}^{\mathcal{I}} \quad [31]$$

where

$$\boldsymbol{C}_{\mathcal{S}}^{\mathcal{I}} = \left[\boldsymbol{C}_{\mathcal{I}}^{\mathcal{S}}\right]^{-1} \quad \text{and} \quad \boldsymbol{C}_{\mathcal{I}}^{\mathcal{S}} = \frac{1}{r} \boldsymbol{C}_{\mathcal{I}}^{E} \quad [32]$$

with

$$\boldsymbol{C}_{\mathcal{I}}^{E} = \begin{bmatrix} \cos(\nu) & \sin(\nu) & 0 \\ -\sin(\nu) & \cos(\nu) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [33]$$

where ν is the true anomaly of the Moon's orbit around the Earth and r is the radius. The detailed derivation can be found in [32].

For our specific case, the Sun's position in the synodic frame is:

$$\boldsymbol{r}_{\odot}^{\mathcal{S}} = \frac{1}{L} \boldsymbol{C}_{\mathcal{I}}^{\mathcal{S}} \boldsymbol{r}_{\odot}^{\mathcal{I}} \quad [34]$$

from which the direction can be computed as:

$$\boldsymbol{d}_{\odot}^{\mathcal{S}} = \frac{\boldsymbol{r}_{\odot}^{\mathcal{S}}}{||\boldsymbol{r}_{\odot}^{\mathcal{S}}||} = \begin{bmatrix} d_x & d_y & d_z \end{bmatrix} \quad [35]$$

Then, Eq. 26 can be applied to obtain the Sun direction in the Blender reference frame.

4. Training and Testing Results

In this section, we present the test results for the analyzed scenario. Two distinct cases are evaluated. First, we address the validation of both the environment and the reward function formulation using state-based observations. After this validation, we then present the results for the image-based observation case. In both cases, the initial conditions for the state are sampled from a uniform distribution over points on the Halo orbit. These sampled conditions are then perturbed by adding random noise. The noise is drawn from a uniform distribution within ±30 km for position and ±0.3 m/s for velocity, simulating realistic deviations from the nominal orbit. The nominal initial mass is equal to 1000 kg. One episode is associated with a complete orbit, and it is divided into 100 steps. The CNN in Fig. 2 is built of three filters having the characteristics in Table 1. The rest of this section will present the two test scenarios and the corresponding results in detail.

Table 1. CNN Architecture.

Layer	Filters	Kernel Size	Padding	Stride
1	16	8×8	4	4
2	32	6×6	3	3
3	128	11×11	0	1

4.1 Case 1: State-based position correction

In this case, the NN consists of just the MLP layers and is fed with the normalized difference between the esti-

mated state and the real one as input instead of the image:

$$\boldsymbol{y}_h = \left[\frac{\hat{\boldsymbol{r}}_h^- - \boldsymbol{r}_h}{||\delta \boldsymbol{r}_{0,\max}||}, \quad \frac{\hat{\boldsymbol{v}}_h^- - \boldsymbol{v}_h}{||\delta \boldsymbol{v}_{0,\max}||} \right] \tag{36}$$

with $\delta \boldsymbol{x}_{0,\max} = [\delta \boldsymbol{r}_{0,\max}, \delta \boldsymbol{v}_{0,\max}]$ being the initial random error to the nominal initial state on the Halo. This observation was selected because it provides the network with all the essential information required for effective learning. The input essentially mirrors the desired output, suggesting that the learning process should be straightforward, provided that the environment and reward functions are properly formulated.

As mentioned in Sec. 3.2, the velocity $\hat{\boldsymbol{v}}_h^+$ is defined as $\hat{\boldsymbol{v}}_h^+ = \boldsymbol{v}_h + \delta \hat{\boldsymbol{v}}$, where $\delta \hat{\boldsymbol{v}}$ represents a random error with a magnitude of approximately 0.3 m/s for each velocity component.

The reward hyperparameters for this case are enlisted in Tab. 2.

δ_{stop}, [km]	w_{cont}	w_{pos}	$w_{penalty}$	ϵ_r, [km]
500	1	1	10	6

Table 2. Reward function hyperparameters.

Figure 5 shows the results obtained during the training process. The reward starts at approximately -0.6 and converges to zero rapidly and with very few oscillations. A similar trend is observed in the episode length, where the curve starts at around 60 and quickly reaches 100 after a few iterations. The mean position error achieves the target within fewer than 1,000 iterations. However, the behavior of the mean position adjustment is particularly noteworthy. Initially, the adjustment drops to small values as the system approaches the target. Once the target is reached, the adjustment increases again, eventually stabilizing at around 9 km. This is driven by the reward component related to correction (as per the second condition in Equation [24]), which incentivizes larger adjustments once the target distance has been achieved.

Overall, these results—with an increasing reward, target-level position error, and non-zero adjustment—indicate that the training was successful.

At this point, the policy was evaluated over 100 different trajectories, each generated with varying initial conditions for position and velocity.

Figure 6 presents the navigation error in position for all 100 trajectories, along with the corresponding 3σ bounds. The error at the first step is larger than at subsequent steps, with a value around 20 km. This initial spike is attributed to the perturbations applied to the initial conditions at the start of each episode. However, despite the larger initial error, it quickly decreases within a few steps, bringing all three position components within the target region.

Based on these evaluation results, both the reward function and environment formulation can be considered well-defined and validated.

4.1.1 Kalman filter for velocity estimation

As introduced in the previous section, the training process incorporated perturbations on the true velocity to simulate its estimation, replicating the presence of a dedicated velocity navigation system.

A simple Extended Kalman Filter (EKF) was implemented and used during evaluation to validate this. Specifically, after the network outputs the correction to estimate the spacecraft's position, this position is provided as an observation to the EKF, which then estimates the velocity at the same time step. This allows the entire estimated state to be retrieved and used as input for the control network responsible for maintaining the spacecraft's trajectory.

The velocity navigation error results are shown in Figure 7. The navigation error for velocity is defined as $v_{err} = ||\hat{\boldsymbol{v}}_h^+ - \boldsymbol{v}_h||$, where $\hat{\boldsymbol{v}}_h^+$ is the velocity estimated by the Kalman filter. In the first step, the velocity error is significantly large, exceeding the target corresponding to the $\delta \hat{\boldsymbol{v}}$ applied during training. This large initial error is attributed to the greater navigation error in position during the first step and the initialization bias of the Kalman filter. However, the error rapidly decreases from the second step onwards and remains within the target region throughout the orbit.

The position error also behaves differently since the Kalman filter influences the overall spacecraft dynamics. Figure 8 shows the position error when the EKF is integrated into the system. As observed previously, the initial position error is larger than in subsequent steps but decreases rapidly within a few steps. However, in contrast to the earlier case (Fig. 6), the 3σ bounds show a slightly different trend. While the y component of the position error behaves similarly to previous cases, the x and z components exhibit some oscillations. These fluctuations can be attributed to the inclusion of the EKF within the navigation system. Nevertheless, the position error remains within the defined threshold for all three components, demonstrating the system's effectiveness in handling the velocity estimation and its compatibility with position-based correction.

4.2 Case 2: Image-based position correction

With the validation of the environment and reward formulation, training with image-based inputs was initiated. The network architecture is depicted in Figure 2.

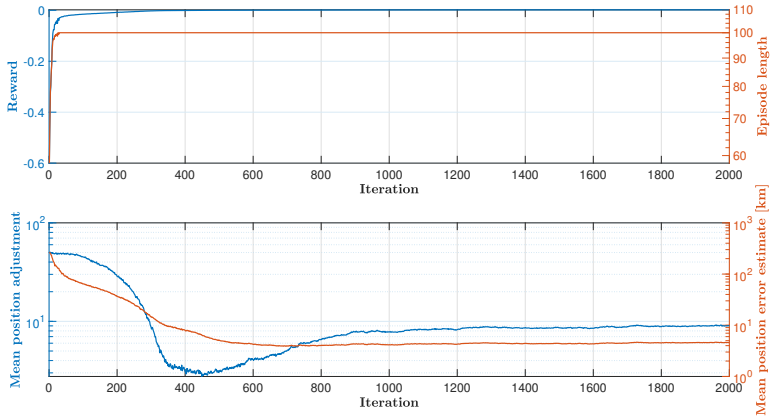

Fig. 5. Training variables progress.

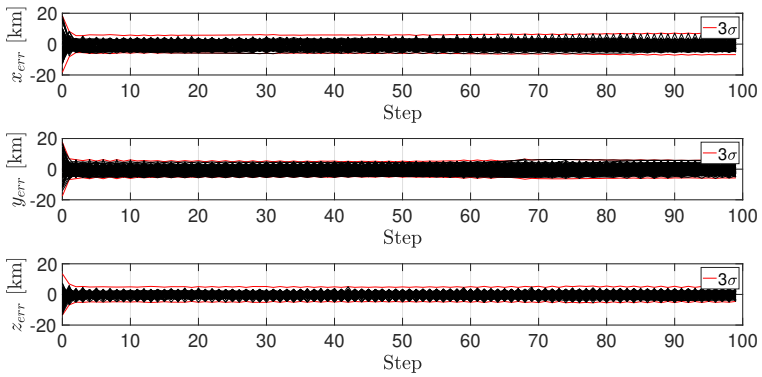

Fig. 6. Navigation error for each position component with associated 3σ bounds.

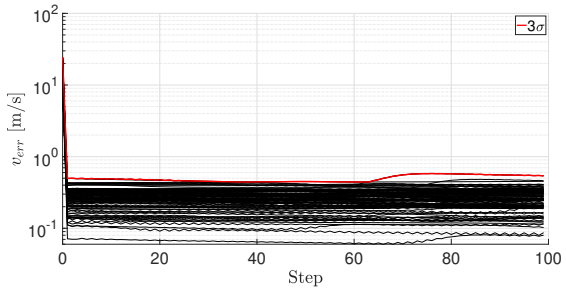

Fig. 7. EKF velocity error.

Following the validation obtained from the position-based training with EKF, the velocity estimation in this case was also simulated by adding a random error of approximately 0.3 m/s to each velocity component.

The evaluation followed the same approach as in previous cases, testing the policy across 100 different trajectories, each initialized with distinct position and velocity conditions.

Figure 9 illustrates the navigation error in position for all the trajectories, along with the corresponding 3σ bounds. Unlike the behavior observed in Figure 6, where a high initial error is followed by stable, lower errors, the trends here exhibit more chaotic variations throughout the episode. Despite this increased variability, the data does not show any signs of divergence, indicating the policy's ability to maintain the errors within acceptable bounds.

These error bounds are further detailed in Table 3. In the table, $\bar{d}_{max}$ represents the maximum distance from the true position among the three position components, averaged over the entire orbit, while $\bar{d}_{norm}$ indicates the average deviation from the reference trajectory across all steps in each episode.

The results demonstrate that the policy maintains a po-

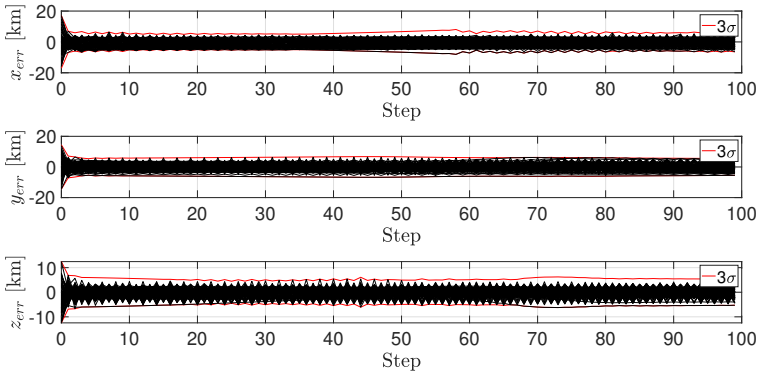

Fig. 8. Navigation error for each position component with the EKF.

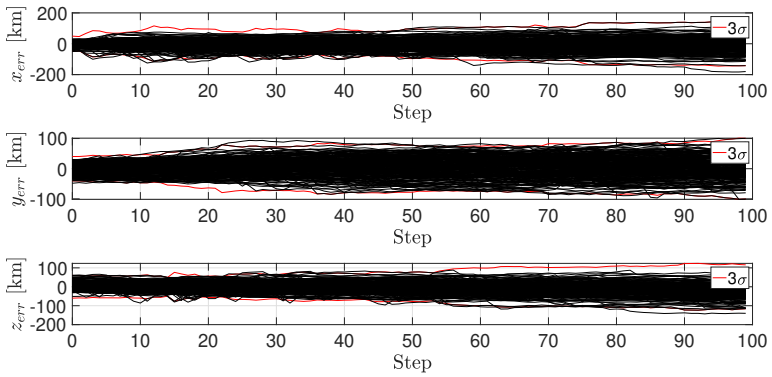

Fig. 9. Navigation error for each position component with associated 3σ bounds.

Table 3. Monte Carlo Simulation Results

$\bar{d}_{max}$, km			$\bar{d}_{norm}$, m/s		
68.3%	95.5%	99.7%	68.3%	95.5%	99.7%
37.71	73.35	104.07	57.33	112.14	153.71

sition navigation error of approximately 150 km in 99.7% of the episodes, with a maximum error on a single component reaching 104 km. Although this error exceeds the originally targeted accuracy, it still represents only 0.2% of the minimum distance between the Halo orbit and the L_2 point. Moreover, it is important to consider that these results were achieved using lower-resolution images that provide less detailed information, introduce more noise, and display smaller distinguishable changes between successive steps relative to the true position. Fig. 10 illustrates the 100 trajectories generated using this method, plotted in cislunar space. As observed, all trajectories closely follow the reference orbit, with minimal deviation, making it challenging to distinguish individual paths from one another. This tight clustering indicates the effectiveness of the navigation system in maintaining proximity to the desired orbit across different initial conditions.

These findings underscore the robustness of reinforcement learning (RL) in managing suboptimal input data. Despite the lower quality of the images, the RL model effectively extracts sufficient information to achieve stable navigation. This highlights the adaptability and potential of RL-based approaches in space navigation tasks. While the system does not achieve highly precise navigation, its ability to function with low-quality data demonstrates the method's resilience and suggests a promising direction for future RL-driven navigation solutions.

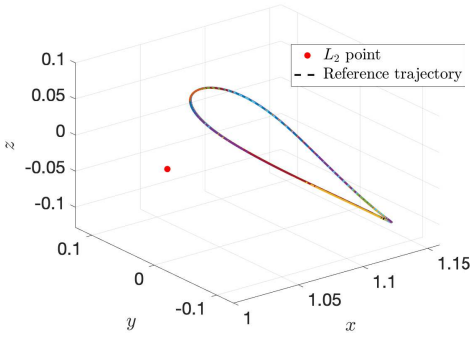

Fig. 10. MonteCarlo results.

5. Conclusions

This study aimed to explore and validate the capability of reinforcement learning (RL) for optical navigation in the cislunar environment. A combination of CNN and MLP is devised to acquire an image (coming from the onboard camera) to generate a correction of the estimated position to perform navigation on a Cislunar Halo orbit. The deep policy network is designed and trained on the distribution of MDPs to enable learning under uncertain conditions. Although this method can be effective, it requires evaluation to operate within the environment distribution adopted in training. Thus, the effectiveness of the navigation policy may deteriorate when deployed outside those bounds. The policy is trained using Proximal Policy Optimization (PPO) with an actor-critic architecture, where both the actor and the critic policies have the same observation. This consists of an image of the Moon modeled using digital terrain models captured from a realistic camera model ray-tracing dependent. This is achieved using a Blender-based simulator connected to the RL framework through a Python Application Programming Interface (API). With this method, the policy can estimate the spacecraft's position with an accuracy of 153 km in 99.7% of the trajectories without presenting any divergence tendency and it can work effectively with a navigation subsystem devised only for velocity estimation. The navigation error obtained corresponds to 0.2% of the closest distance between the L_2 point and the Halo orbit. However, comparing these results with the validation case (Case 1), it can be seen how reducing the quality of the observation affects directly the overall performance of the proposed approach. Indeed, the images carry less information compared to the actual state, making it more difficult for the net to generate a precise output. A key limitation of reinforcement learning (RL) in this context is its reliance on low-quality images. However, the ability to process and extract useful information from low-quality images can also be viewed as a strength. Despite the limited detail in the images, the network consistently produced accurate position corrections, yielding promising results. Nevertheless, the method's greatest strength lies in the near-instantaneous generation of outputs once the policy is trained. Once the policy weights are optimized, deploying the system is computationally efficient, requiring only matrix multiplication with the input, making it suitable for onboard real-time applications. In summary, despite the challenges of long training times and the use of low-resolution images, this application of RL for optical navigation in the cislunar environment can be considered a success. The RL model demonstrated its ability to reliably estimate the spacecraft's position, showcasing the robustness of the approach even with limited observational data.

5.1 Future work

The primary challenge encountered was reinforcement learning's difficulty in handling high-quality images, necessitating the use of low-quality ones. These lower-resolution images lacked sufficient detail, preventing the network from achieving the desired precision. This issue likely arose because successive orbit steps produced images with minimal differences, resulting in similar inputs for distinct outputs, which is difficult for the network to process. Even with navigation errors of hundreds of kilometers, the images did not exhibit significant variations, making it hard for the policy to refine the spacecraft's trajectory. Given these challenges, it would be worthwhile to investigate the performance of this RL-based method in a different orbit, such as a transfer orbit between two cislunar Halo orbits, rather than a stable closed-loop orbit. In such a scenario, the images would show more substantial changes at each step, potentially providing the network with more distinct information to improve navigation accuracy. Another possible enhancement involves using a more complex dynamical model for the real-world environment, such as incorporating the Four-Body Problem or a Two-Body Problem with perturbations, while keeping the Circular Restricted Three-Body Problem (CR3BP) model onboard. This approach would better reflect the real-world conditions, where the spacecraft's actual motion differs from the simplified model used in the onboard system. Additionally, to develop a fully RL-based navigation system, an interesting improvement could be the integration of a second policy to estimate velocity based solely on position, eliminating the need for the Extended Kalman Filter (EKF) currently used. This would streamline the system and create a more cohesive RL-based framework for space navigation.

References

[1] E. Turan, S. Speretta, and E. Gill, "Autonomous navigation for deep space small satellites: Scientific and technological advances," *Acta Astronautica*, vol. 193, pp. 56–74, 2022. DOI: https://doi.org/10.1016/j.actaastro.2021.12.030.

[2] C. L. Thornton and J. S. Border, *Radiometric tracking techniques for deep-space navigation*. John Wiley & Sons, 2003.

[3] C. Frost, "Challenges and opportunities for autonomous systems in space," in *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2010 Symposium*, 2010. DOI: https://doi.org/10.17226/13043.

[4] R. P. De Santayana and M. Lauer, "Optical measurements for rosetta navigation near the comet," in *Proceedings of the 25th International Symposium on Space Flight Dynamics (ISSFD), Munich*, 2015.

[5] F. Terui *et al.*, "Guidance, navigation, and control of hayabusa2 touchdown operations," *Astrodynamics*, vol. 4, pp. 393–409, 2020.

[6] R. Raymond Karimi and D. Mortari, "Interplanetary autonomous navigation using visible planets," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 6, pp. 1151–1156, 2015.

[7] D. Mortari and D. Conway, "Single-point position estimation in interplanetary trajectories using star trackers," *Celestial Mechanics and Dynamical Astronomy*, vol. 128, pp. 115–130, 2017.

[8] V. Franzese and F. Topputo, "Optimal beacons selection for deep-space optical navigation," *The Journal of the Astronautical Sciences*, vol. 67, no. 4, pp. 1775–1792, 2020.

[9] B. Jia and M. Xin, "Vision-based spacecraft relative navigation using sparse-grid quadrature filter," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 5, pp. 1595–1606, 2012.

[10] S. Li, P. Cui, and H. Cui, "Vision-aided inertial navigation for pinpoint planetary landing," *Aerospace Science and Technology*, vol. 11, no. 6, pp. 499–506, 2007.

[11] J. A. Christian, "Optical navigation using planet's centroid and apparent diameter in image," *Journal of guidance, control, and dynamics*, vol. 38, no. 2, pp. 192–204, 2015.

[12] J. A. Christian, "Optical navigation using iterative horizon reprojection," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 5, pp. 1092–1103, 2016.

[13] J. A. Christian and S. B. Robinson, "Noniterative horizon-based optical navigation by cholesky factorization," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 12, pp. 2757–2765, 2016.

[14] S. Silvestrini *et al.*, "Optical navigation for lunar landing based on convolutional neural network crater detector," *Aerospace Science and Technology*, vol. 123, p. 107 503, 2022. DOI: https://doi.org/10.1016/j.ast.2022.107503.

[15] S. Sharma, C. Beierle, and S. D'Amico, "Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks," in *2018 IEEE Aerospace Conference*, IEEE, 2018, pp. 1–12.

[16] P. F. Proença and Y. Gao, "Deep learning for spacecraft pose estimation from photorealistic rendering," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 6007–6013.

[17] D. Hirano, H. Kato, and T. Saito, "Deep learning based pose estimation in space," in *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Madrid, Spain*, 2018, pp. 4–6.

[18] S. C. Stubberud, R. N. Lobbia, and M. Owen, "An adaptive extended kalman filter using artificial neural networks," in *Proceedings of 1995 34th IEEE Conference on Decision and Control*, IEEE, vol. 2, 1995, pp. 1852–1856.

[19] A. G. Parlos, S. K. Menon, and A. F. Atiya, "Adaptive state estimation using dynamic recurrent neural networks," in *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, IEEE, vol. 5, 1999, pp. 3361–3364.

[20] D.-J. Jwo and C.-F. Pai, "Incorporation of neural network state estimator for gps attitude determination," *The Journal of Navigation*, vol. 57, no. 1, pp. 117–134, 2004.

[21] G. Revach, N. Shlezinger, X. Ni, A. L. Escoriza, R. J. Van Sloun, and Y. C. Eldar, "Kalmannet: Neural network aided kalman filtering for partially known dynamics," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1532–1547, 2022.

[22] M. Pugliatti and F. Topputo, "Design of convolutional extreme learning machines for vision-based navigation around small bodies," *arXiv preprint arXiv:2210.16244*, 2022.

[23] L. Federici *et al.*, "Image-based meta-reinforcement learning for autonomous guidance of an asteroid impactor," *Journal of Guidance, Control, and Dynamics*, vol. 45, no. 11, pp. 2013–2028, 2022. DOI: https://doi.org/10.2514/1.G006832.

[24] L. Federici, A. Scorsoglio, A. Zavoli, and R. Furfaro, "Meta-reinforcement learning for adaptive spacecraft guidance during finite-thrust rendezvous missions," *Acta Astronautica*, vol. 201, pp. 129–141, 2022. DOI: 10.1016/j.actaastro.2022.08.047.

[25] A. Scorsoglio, A. D'Ambrosio, L. Ghilardi, B. Gaudet, F. Curti, and R. Furfaro, "Image-based deep reinforcement meta-learning for autonomous lunar landing," *Journal of Spacecraft and Rockets*, vol. 59, no. 1, pp. 153–165, 2022. DOI: https://doi.org/10.2514/1.A35072.

[26] L. Ghilardi, A. D'Ambrosio, A. Scorsoglio, R. Furfaro, and F. Curti, "Image-based lunar landing hazard detection via deep learning," in *Proceedings of the 31st AAS/AIAA Space Flight Mechanics Meeting, Virtual*, 2021, pp. 1–4.

[27] J. Broida and R. Linares, "Spacecraft rendezvous guidance in cluttered environments via reinforcement learning," in *29th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Society, 2019, pp. 1–15.

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

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

[30] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[31] C. La Pegna, L. Federici, A. Zavoli, and R. Furfaro, "Meta-reinforcement learning for adaptive station-keeping in cislunar periodic orbits," in *2023 AAS/AIAA Astrodynamics Specialist Conference*, Big Sky, MT, 2023.

[32] N. Assadian and S. H. Pourtakdoust, "On the quasi-equilibria of the bielliptic four-body problem with non-coplanar motion of primaries," *Acta Astronautica*, vol. 66, no. 1-2, pp. 45–58, 2010.