Post-Print

This is the accepted version of:

L. Capra, A. Brandonisio, M. Lavagna
Network Architecture and Action Space Analysis for Deep Reinforcement Learning Towards Spacecraft Autonomous Guidance
Advances in Space Research, Published online 01/12/2022
doi:10.1016/j.asr.2022.11.048

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Advances in Space Research xx (2022) xxx-xxx

ADVANCES IN SPACE RESEARCH
(a COSPAR publication)

www.elsevier.com/locate/asr

Network architecture and action space analysis for deep reinforcement learning towards spacecraft autonomous guidance

Lorenzo Capra[a,*], Andrea Brandonisio[a], Michèle Lavagna[b]

[a] PhD Student, Aerospace Department, Politecnico di Milano, Milan, Via La Masa 24, 20156 Italy
[b] Full Professor, Aerospace Department, Politecnico di Milano, Milan, Via La Masa 24, 20156 Italy

Abstract

The growing ferment towards enhanced autonomy on-board spacecrafts is driving the research of leading space agencies. Concurrently, the rapid developments of Artificial Intelligence (AI) are strongly influencing the aerospace researches, regarding on-orbit servicing (OOS) activities above all. Within the wide spectrum of OOS and proximity operations, this work focuses on autonomous guidance of a chaser spacecraft for the map reconstruction of an artificial uncooperative target. Adaptive guidance is framed as an active Simultaneous Localization and Mapping (SLAM) problem and modeled as a Partially Observable Markov Decision Process (POMDP). A state-of-the-art Deep Reinforcement Learning (DRL) method, Proximal Policy Optimization (PPO), is investigated to develop an agent capable of cleverly planning the shape reconstruction of the uncooperative space object. The guidance algorithm performance are evaluated in terms of target map reconstruction, by rendering the space object with a triangular mesh and then considering the number of quality images for each face. A major differentiation in the algorithm implementation is provided by the employment of either a discrete or a continuous action space. The main differences between the two cases are critically commented and the benefits of a continuous action space are highlighted. The proposed model is trained and then extensively tested, always starting from random initial conditions, to verify the generalizing capabilities of the DRL agent, by means of the neural network architecture. On this note, a comparison analysis between a Feed-forward Neural Networks (FFNN) and a Recurrent Neural Network (RNN) is performed. The better performing model is retrieved from the aforementioned comparisons, and its robustness and sensitivity are sharply analyzed. This work confirms and develops further the applicability of DRL techniques for autonomous guidance, highlighting in a critical way its possible implementation in future close proximity scenarios.

Keywords: on-orbit servicing; relative dynamics; reinforcement learning; imaging; shape reconstruction

1. Introduction

Spacecraft autonomy is one of the most critical problems that nowadays are driving the research of leading space agencies, since an enhanced spacecraft independence would allow for reliable, cost-effective, lower risk services, and for much more flexible missions. Autonomous flight operations are particularly attractive in the context of on-orbit servicing (OOS) activities (Tatsch et al., 2006), which refers to all those operations in space that a *servicer* conducts on another resident space object (RSO), called *client*. The client may then be *cooperative*, meaning that it provides useful information to the servicer directly, or to ground-stations, thus facilitating the operations, or it could be *non-cooperative*, when it does not share any significant information with the servicer. Among OOS activities, the spectrum of proximity operations studies has constantly grown in the last decade, but not even remotely at the same rate of modern Machine Learning (ML), and specifically Reinforcement Learning (RL). The field of Artificial Intelligence (AI) is

*Corresponding author: Tel.: +39-333-824-7339;
Email addresses: lorenzo.capra@polimi.it (Lorenzo Capra), andrea.brandonisio@polimi.it (Andrea Brandonisio), michelle.lavagna@polimi.it (Michèle Lavagna)

https://dx.doi.org/10.1016/j.jasr.xxxx.xx.xxx
0273-1177/

moving a lot faster than space research, so the presented work aims at shrinking this gap, by applying state-of-the-art RL techniques to understand the feasibility of these different mathematical approaches, with the aim to improve spacecraft autonomy.

In the last years, several AI tools have been applied in the Guidance, Navigation and Control (GNC) chain, to help automatizing some spacecraft operations or exploiting higher performances in classical problems. Concerning the navigation technology, for example, machine learning have greatly pushed the development of vision-based techniques especially for lunar landing problem. Emami et al. (2019) and Downes et al. (2020) exploit convolutional neural networks (CNN) to detect lunar craters in camera images. Furfaro et al. (2018) exploit deep learning to automatize the landing, while Silvestrini et al. (2022), taking advantage of CNN crater detector, design an absolute lunar landing navigation algorithm. Some very interesting and promising results have been achieved also exploiting neural networks to enhance the spacecraft guidance and control in relative motion scenarios for safe autonomous maneuvers between formation flight satellites (Silvestrini & Lavagna, 2021b), and distributed system reconfiguration (Silvestrini & Lavagna, 2021a). In (Sullivan & Bosanac, 2020), reinforcement learning is exploited to design, with low-thrust propulsion, a multi-body system transfer trajectory. Instead, the first formulation and design of a spacecraft guidance as a Partially Observable Markov Decision Process (POMDP) problem was proposed by Pesce et al. (2018) and then developed further by Chan & Agha-mohammadi (2019) and Piccinin et al. (2022), who adopted Reinforcement Learning (RL) to plan the trajectory of a chaser spacecraft for small-bodies imaging. Major contribution to the application of RL techniques is given by Gaudet et al. (2020a), Gaudet et al. (2020b) and Hovell & Ulrich (2021), where planetary landing and close proximity operations are investigated. In Brandonisio et al. (2021), DRL techniques were firstly exploited for the shape reconstruction of artificial objects.

Within this context, this and its previous works want to develop an innovative approach for spacecraft trajectory path-planning around uncooperative and unknown space objects, for the shape and map reconstruction in a relative motion scenario. Indeed, the spacecraft shall autonomously explore the surrounding environment and plan the following actions to take. Thus, the problem falls in the *active* Simultaneous Localization and Mapping (SLAM) (Durrant-Whyte & Bailey, 2006) framework, since the planning operations is also performed. SLAM may be phrased as a POMDP, which entails an agent interacting with the environment and exchanging information with it. The goal is to solve for the decision-making policy of the agent, and to do so Deep Reinforcement Learning (DRL) techniques are employed. Reinforcement Learning (RL) algorithms are a powerful tool when dealing with decision-making problems and the combination with Neural Networks (NN) in DRL allows to improve the generalizing capabilities of the resulting policy, and to solve more complex problems characterized by high-dimensional, continuous state and action spaces and partial observability (Sutton & Barto, 2018). This approach is preferred to fuzzy logic or evolutionary algorithm, especially because DRL and NNs are more suitable for prediction problems, and in dealing with continuous data environment. Moreover, the capability of neural networks to generalize their behaviour and follow non prescribed rules has been considered determinant for the methodology choice. Brandonisio (2019-2020) and Capra (2020-2021) are regarded as reference points for this study and the aim of this work is to take a step forward in the autonomous mapping of uncooperative artificial space objects problem, advancing some of the current research limitations, exploring more complex network architectures, juxtaposing different action space dimensions of the same method, confirming and validating the applicability of DRL techniques in such context. A state-of-the-art Deep Reinforcement Learning algorithm, Proximal Policy Optimization (PPO), developed by Schulman et al. (2017), is investigated to solve for the chaser decision-making policy, mapping the input observations to the output action to take. Extensive training and testing campaigns are carried out, to verify the models used and the obtained results, comparing different implementations performance level. The problem architecture has already proved to be feasible in Brandonisio (2019-2020), therefore this work wants to analyze directly the DRL performance, in terms of algorithm implementation and neural networks model. In particular, this paper is focused on the examination of two different neural network models, feed-forward (FFNN) and recurrent (RNN), aiming to understand the advantages and disadvantages of both in terms of training performance and stability. Moreover, this work also deeply analyses the DRL action space distinguishing between agents employing either discrete or continuous action spaces, as developed by Capra (2020-2021). To complete the study, the main DRL model is tested in order to assess the robustness and sensitivity of the trained agent to unseen conditions and scenarios.

1.1. Paper Overview

The sections of this work are structured as follows: in Sec. 2, the overall tool architecture and scenario are presented; in Sec. 3, the autonomous guidance problem is defined as a Partially Observable Markov Decision Process (POMDP); in Sec. 4, the DRL algorithm used is presented and explained, while Sec. 5, declines the problem in the different DRL actors. After this overview, useful to understand the base of the work, in Sec. 6 and Sec. 7, the two paper pillars, the comparison between different network models and algorithm action space models are presented. At last, in Sec. 8 some robustness and sensitivity analysis performed on the trained model are discussed.

2. Problem statement

The goal of this research is the autonomous path-planning strategy for the shape reconstruction of an uncooperative and unknown space object, in a close proximity relative motion scenario. The development of an autonomous guidance algorithm depends on the overall GNC architecture, which is described in Fig. 1. Note that the image processing and pose estimation block are out of the scope of this work, which considers their outputs as the necessary information for the development of the autonomous guidance algorithm. As such they have not been implemented.

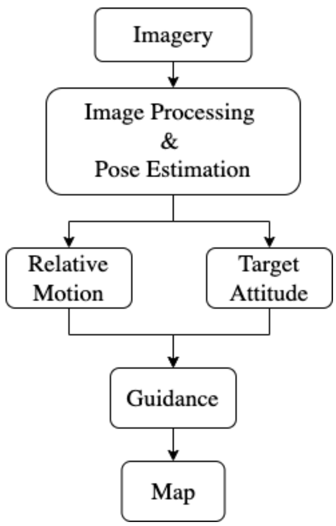


Fig. 1: Fly-around planning architecture.

The problem scenario is defined to have as inputs to the guidance block the relative motion between the chaser and the target and the attitude of the uncooperative object. These may come from image processing and pose estimation techniques, which may work with a vision-based system. This could be developed by either implementing visible-only (VIS) imagery, or by employing both visible (VIS) and thermal infrared (TIR) imaging to improve the navigation accuracy. The latter has been proposed by Civardi et al. (2021) in the framework of small-bodies, and it demonstrated the effectiveness of combining imagery from different bands. This allows to avoid problems of illumination condition typical of VIS-only systems. The work here developed considers and discusses the RL formulation in both cases, as will be deeply described in Sec 5.3. The implementation of the navigation system is out of the scope of the presented work, that only focuses on the development of the guidance algorithm; therefore, the information regarding relative motion and target attitude will be assumed to be known at each step. The image acquisition, for this particular problem statement, is not only needed by the navigation block, but can also be used to reconstruct the shape of the unknown object. In order to do so, different techniques can be considered, such as stereophotoclinometry (SPC) developed by Gaskell (2001). To correctly define the problem, the target surface is subdivided into maplets, via triangular mesh, and a visibility model is defined to constantly compute the relative orientation between the cameras on the chaser and the target to understand which faces are illuminated (if necessary) or in the cameras field of view (FOV). With a sufficient number of images for each face, the target map is considered complete. A better clarification on how the map is reconstructed is given in Sec. 5.3.

So, the overall objective of the problem, independently from the resolution strategy adopted, is a spacecraft that autonomously plans the trajectory to be followed to efficiently reconstruct the shape of the uncooperative object. Machine learning, and specifically Reinforcement Learning, can be exploited to model the guidance block and solve for the spacecraft decision-making behavior, taking advantage of all its benefits, that are discussed in the next sections.

3. Autonomous guidance

The autonomous exploration and trajectory planning in an unknown environment is formulated as an active Simultaneous Localization and Mapping (SLAM) problem, in which an agent builds a map of its surroundings while concurrently estimating its positions and planning the next actions to take. These problems can be phrased as a Partially Observable Markov Decision Process (POMDP) (Kurniawati, 2021). The next section aims at developing the mathematical tools necessary to understand the problem and how it is solved.

3.1. Partially Observable Markov Decision Process

A Markov Decision Process (MDP) is a problem formulating an agent decision making in a stochastic and sequential environment. The essence of the model is that the agent inhabits an environment that changes accordingly to the actions taken, and the state of this environment affects the reward signal as well as the probability to transition to a certain new state. A POMDP is a MDP with state uncertainty, meaning the agent cannot know the *true* state, but only a *belief* state using observations. This formulation is valid whenever the agent senses the environment via on-board sensors, which inherently introduce errors in their measurements, or when it may not be able to observe all the state variables describing the environment.

A POMDP is characterized by a 6-tuple ($\mathbf{S}$, $\mathbf{A}$, $\mathbf{R}$, $\mathbf{T}$, $\mathbf{\Omega}$, $\mathbf{O}$):

- $\mathbf{S}$ is the space of all possible states s in the environment;
- $\mathbf{A}$ is the space of all possible actions a that can be taken in all the states of the environment;
- $\mathbf{R}$ is the reward function, guiding the action selection to maximize it;
- $\mathbf{T}$ $(s_{k+1}|\, s_k, a_k)$ is the transition function governing the probability of moving from one state to the next, given the current state and an action at timestep k;
- $\mathbf{\Omega}$ is the space of possible observations;
- $\mathbf{O}$ $(o_{k+1}|\, a_k, s_{k+1})$ is the probability of making a particular observation, taking an action that leads to a particular new state.

This type of problems is quite complex to solve and may become computationally intractable if not reduced to a simpler MDP. This can be done including the history h, that plays the role of an archive of past actions and observations. The new formulation, known as belief-space MDP, is described by a 4-tuple ($\mathbf{B}$, $\mathbf{A}$, $\mathbf{R}$, $\mathbf{T}$):

- **B** is the belief space, where the *belief* is defined as $b = p(s|h)$, so it is the probability of being in a certain state s after the history h.

Solving a POMDP means computing a *policy* π, which represents the mapping function from states s to actions a that the agent is employing at each step k. This decision-making feature is said to be "optimal" if the agent concurrently maximizes the reward function, which mathematically expresses the problem objectives. Thus, maximizing the reward signal received is equivalent to reaching the goal set by the designer, depending on the problem at hand. In case of an infinite horizon problem, the optimal policy is defined as in Eq. 1:

$$\pi_* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(a_k, b_k) \right] \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor, introduced as a mechanism to control how myopic or short-sided is the agent, exponentially decaying the effect of rewards far away in time. R represents indeed the reward signal, depending on the action a and belief b at step k.

Considering the proposed work, a direct link between the elements describing a POMDP and the autonomous guidance problem characters can be highlighted:

- the agent is the chaser spacecraft, interacting with the surrounding environment, governed by the problem dynamics and visibility model, discussed in Sec. 5;
- the belief space b is computed from the sensors measurements, passed to the image processing and pose estimation steps;
- the action space a is the result of the available actuators activity, such as the force exerted by switching on some spacecraft thrusters;
- the reward function strictly depends on the objectives, so in this case the shape reconstruction of the uncooperative target.

4. Deep Reinforcement Learning

Reinforcement Learning is a widely employed tool for solving MDPs (Sutton & Barto, 2018), and its combination with Neural Networks, pioneered by Mnih et al. (2015), for function approximation, allows to solve many complex problems characterized by high-dimensionality and partial observability. A state-of-the-art Deep Reinforcement Learning algorithm, Proximal Policy Optimization (PPO) (Schulman et al., 2017), is examined to solve for the spacecraft decision-making policy.

4.1. Proximal Policy Optimization

PPO is a policy-gradient method, belonging to the Actor-Critic family (Mnih et al., 2016). It outclasses most of the other DRL algorithms in many typical benchmark problems, because of its improved training stability. It builds up from the Trust Region Policy Optimization (TRPO) method (Schulman et al., 2015), retaining its reliability and data efficiency, but then handles the loss function in a much simpler and well-planned fashion. Starting from the TRPO loss function, which exploits the probability ratio, as in Eq. 2, between the policy π_w at two subsequent timesteps, PPO increases training robustness by clipping the objective function and limiting the possible update, so that the policy does not change drastically. The simple expression for the PPO loss function $L^{CLIP}(w)$ is reported in Eq. 3.

$$p_k(w) = \frac{\pi_w\,(a_k|s_k)}{\pi_w\,(a_{k-1}|s_{k-1})} \tag{2}$$

$$L^{CLIP}(w) = \hat{\mathbb{E}}_k\, \left[min\,(p_k(w)\,,\, clip\,(p_k(w), 1-\epsilon, 1+\epsilon)) \right] A_k \tag{3}$$

In both Eq. 2 and Eq. 3, w refers to the networks parameters (i.e. their weights and biases). The parameter ϵ indicates the clipping factor, while A_k is the advantage function, retained from the Advantage Actor Critic (A2C) (Mnih et al., 2016) formulation and representing how better a selected action is compared to all the others at a given state. It is simply computed as the difference between the discounted reward signal r and the state value function V, computed by the critic network and depending on the current state s_k. This concept is described in Eq. 4.

$$A(s_k, a_k) = \left[\sum_{j=k}^{T} \gamma^{j-k} r(s_k, a_k) \right] - V(s_k) \tag{4}$$

In Eq. 3 the clipping function *clip* limits the probability ratio to be inside the range define by $1+\epsilon$ and $1-\epsilon$. Therefore, thanks to the objective function clipping, multiple epochs of gradient descent can be run on the sample data without causing destructively large policy updates, and squeezing every ounce of information it can learn from.

Common practice in PPO algorithms is the addition of an entropy regularization term multiplying the state s_k to the clipping objective function, as in Eq. 5, to ensure a sufficient exploration level during training:

$$L^{PPO}(w) = L^{CLIP}(w) + c_2 S(\pi_w) s_k \tag{5}$$

where $S(\pi_w)$ is the entropy bonus term, that is function of the current policy, and c_2 a scalar multiplying factor that determines the influence of the entropy term on the overall loss function.

The critic network is itself trained by means of optimizing a simple mean squared error (MSE) objective function, defined in Eq. 6.

$$L_{critic} = \sum_{i=1}^{N} \left(V(s_k^i) - \left[\sum_{j=k}^{T} \gamma^{j-k} r(s_j^i, a_j^i) \right] \right)^2 \tag{6}$$

To avoid confusion, in the previous equation, the subscript k stands for the considered time-step instant; while the superscript i stands for the current batch used for the loss function computation. Regarding the practical implementation of the method, a number of hyperparameters need to be introduced and detailed. For an in depth description of all the parameters refer to the original PPO paper (Schulman et al., 2017). The main ones are now discussed:

- the discount factor γ is the one presented in Sec. 3.1, ruling how farsighted is the agent;
- the Generative Adversarial Estimator λ also contributes to reward shaping;
- the clipping factor ϵ corresponds to the acceptable threshold of divergence between the old and new policies during gradient descent updating. Setting this value to a small number will result in more stable updates, but will also slow the training process;
- the entropy coefficient β acts as a regularizer and prevents premature convergence which in turn may prevent sufficient exploration;
- the batch size corresponds to how many experience timesteps are used for each gradient descent update;
- the buffer size corresponds to how many experiences should be collected before gradient descent is performed on them all. This should be a multiple of the batch size, otherwise a batch is truncated and may poorly affect the optimization step. Typically larger buffer sizes correspond to more stable training updates;
- the number of epochs is the number of passes through the experience buffer during gradient descent. The larger the batch size, the larger it is acceptable to make this. Decreasing this will ensure more stable updates, at the cost of slower learning.

A brief analysis of the algorithm working principles is now commented and the pseudo-code for the PPO algorithm is presented in Algorithm 1.

Algorithm 1 Proximal Policy Optimization

1: Initialize actor and critic networks parameters w_0, ϕ_0
2: Initialize batch
3: **while** batch step $i \leq$ batch size **do**
4: Collect set of trajectories T_k by running policy $\pi_k = \pi(w_k)$ in the environment
5: Compute rewards R_k
6: Compute Advantages A_k based on the current value V_k
7: **end while**
8: Compute the probability ratio $p_k(w_k)$
9: Update the policy by maximizing the clipped objective function via stochastic gradient descent
10: Update the value function via regression on MSE

The algorithm loops until it has collected a certain batch size of data, obtained through the interaction of the agent with the environment. At each step it stores a set of observations, actions taken by the agent, rewards output by the surrounding environment and the advantages values. Once it retrieves a number of transitions specified by the batch dimension of the experience buffer, the probability ratio and consequently the loss functions for both the actor and the critic are computed as in Eq. 2, Eq. 3 and Eq. 6. With these losses the two neural networks are then updated via backpropagation with an *Adam* optimizer. This way the networks adjust their parameters to better fit the problem objectives.

5. Reinforcement Learning framework

The work proposes an innovative decision-making process to autonomously plan the pseudo-optimal guidance around an uncooperative and unknown space object through Deep Reinforcement Learning. This is coupled with a pre-processing phase, representing the *navigation* part of a GNC algorithm, in which information coming from the external environment, sensors and the object conditions are elaborated to estimate the state. It is worth to underline that the navigation process is given for granted, directly outputting the state information. This is then fed to the autonomous guidance agent which crafts the control policy to maximize the reward, affecting the environment and all the others information providers. In the next sections, a detailed and critical description of all the architecture components is presented, according to a DRL framework. Indeed, three main characters emerge from the decision-making problem just reviewed: the state, the agent policy, and the reward function.

5.1. State space model

The *state space* model is the set of information coming from the environment. For the scope of this work, it is assumed that the agent perfectly knows the state variables at each timestep, while in practice these data are measured with sensors, inherently introducing precision errors. The state vector fed to the agent should be tailored in such a way that it contains only essential information for the decision-making process, to build a policy capable of selecting the appropriate action in every condition the agent may find itself. Eq. 7 defines the state model used for this work.

$$\mathbf{S} = \begin{Bmatrix} d \\ v \\ \theta \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \theta_x \\ \theta_y \\ \theta_z \\ \dot{\theta}_x \\ \dot{\theta}_y \\ \dot{\theta}_z \end{Bmatrix} \tag{7}$$

Two main factors have been considered in order to identify the state space information: the possibility of estimating these quantities by means of on-board instruments (fundamental for an autonomous spacecraft) and the compatibility of these data to ease the agent learning in identifying the close proximity scenario. The vector in Eq. 7 contains the relative motion (position and velocity vectors d and v) between the chaser and the target, together with the attitude information about the uncooperative space object, in terms of angles θ and angular velocities $\dot{\theta}$. The selected variables are exactly the ones that describes the relative pose between the two objects, under the assumption of the chaser spacecraft always pointed towards the object. As a consequence, this state dictates how the surrounding environment changes over time.

The orbital dynamics of the system, describing the relative translational motion between the spacecraft and the object, is based on the linearized eccentric model proposed by Tillerson Inalhan et al. (2002), reported in Eq. 8 in the Local Vertical Local Horizontal (LVLH) reference frame centered in the target:

$$\begin{cases} \ddot{x} = \frac{2\mu}{r^3}x + 2\omega\dot{y} + \omega^2 x + a_x \\ \ddot{y} = \frac{-\mu}{r^3}y - 2\omega\dot{x} - \omega^2 y + a_y \\ \ddot{z} = \frac{-\mu z}{r^3} + a_z \end{cases} \tag{8}$$

where r in this case is the radius of the target orbit, μ is the primary attractor gravitational parameter, and $\omega = \dot{f}$, in Eq. 9, is the time derivative of the target true anomaly and it is expressed as follows:

$$\omega = \dot{f} = \frac{n(1 + e\cos f)^2}{(1 + e^2)^{\frac{3}{2}}} \tag{9}$$

with f being the target true anomaly, e its orbit eccentricity, and $n = \sqrt{\frac{\mu}{r^3}}$ the mean motion.

In particular, note the relative motion between the two objects is directly affected by the agent actions that influence the set of equations by means of an acceleration vector $[a_x, a_y, a_z]$.

As for the target object attitude dynamics, Euler equations are used, assuming the small angles approximation and expressing them in the LVLH frame, as in Eq. 10.

$$\begin{cases} I_x\ddot{\theta}_x + n(I_z - I_y - I_x)\dot{\theta}_y + n^2(I_z - I_x)\theta_x = 0 \\ I_y\ddot{\theta}_y + n(I_x + I_y - I_z)\dot{\theta}_x + n^2(I_z - I_x)\theta_y = 0 \\ I_z\ddot{\theta}_z = 0 \end{cases} \tag{10}$$

with I_x, I_y, I_z being the principal components of inertia of the target.

This equation refers specifically to the target rotational motion, which is exploited to retrieve the orientation of the object mesh faces at each time step. Then, from the normal vector to each face, it is possible to evaluate which parts of the target are in visibility of the cameras and consequently build the map. More on this is later explained in Sec. 5.3.

As already underlined, please note that this work is based on the main assumption of spacecraft cameras always pointed towards the target center of mass. Therefore, the chaser attitude dynamics is neglected, simplifying the formulation of the already quite complex problem. A small remark should be made regarding the Euler equations formulation for the target: since the object could ideally be unknown, the necessity of finding its center of mass to place the principal axis frame may be problematic. At the current stage, since the main concern is proving the applicability of the proposed architecture, this assumption seems reasonable, but should be kept in mind when refining further the model.

In conclusion. as explained in Sec. 3.1, the environment is only partially observable, therefore the here defined *state* space corresponds to the POMDP observation space, representing only part of the overall information that would be needed to reconstruct the full environment.

5.2. Agent policy

The agent interacts with the surrounding environment, receiving the state observations and a reward signal and selecting accordingly the action to take. It is characterized by its policy, which governs the decision-making strategy adopted. As explained in Sec. 3.1, the goal is to optimize the policy π, to maximize the reward function. In the next paragraphs the elements defining the agent policy are presented.

Action space model. The action space represents all the possible decisions that the agent could take at each timestep with its policy. Through its action, the agent can interact with the surrounding environment, entering the equations of motion in Eq. 8 by means of an acceleration vector coming from the thruster.

This section encapsulates one of the main dichotomies with respect to the agent decision-making strategy. Indeed, depending on its dimensionality, the action space can be either *discrete* or *continuous*. In the former case, the action is selected among a predefined set of possible fixed thrust impulses. In the second case, instead, the control action is directly the acceleration vector. One of the main goals of this work is to compare the two possible action spaces, in terms of performance and stability.

In the *discrete* case, the action is selected between the predefined thrust impulses fixed both in direction and magnitude, defined in Eq. 11.

$$A = \left[+T_x,\ -T_x,\ +T_y,\ -T_y,\ +T_z,\ -T_z,\ 0 \right] \tag{11}$$

where $+T_x$ represents an impulsive maneuver along the positive direction of the spacecraft x-axis, $-T_x$ represents an impulsive maneuver along the negative direction of the spacecraft x-axis; the same is applicable for the other components. The impulse maneuver assumes a constant acceleration value equal to $a = 0.001 m/s^2$. At each timestep the actor network chooses the most suitable action with a softmax activation function on the output set in Eq. 11. This is a simpler implementations, which would result in a fast training process, because of the limited options available to the spacecraft. In this case, a cold-gas thrusters propulsion system is considered as baseline, as used in (Brandonisio, 2019-2020).

With a continuous action space, instead, the control action is a tridimensional vector pointing ideally to any direction in space. Since it is continuous, the magnitude of the thrust vector can vary inside the limits specified by the propulsion system on-board. For this analysis, a single thruster electric propulsion (Martínez et al., 2019) is employed and the most notable attributes affecting the action selection are the maximum thrust (T_{max}) and the minimum impulse bit (MIB), since they define the range inside which the decision-making policy can select the magnitude of the action. The actor network in this case samples the three components of the thrust vector from a Gaussian distribution defined by a mean value μ and a standard deviation σ, which is connected to the exploration/exploitation dilemma in RL (Sutton & Barto, 2018) and defines the confidence level of the policy in the selected action. This solution is more realistic, but also more computationally expensive, as the action space dimensionality is practically infinite.

Beyond the space definition, another important parameter for the action model is the control interval Δt, that defines the time elapsing between two subsequent control actions. Setting it entails a trade-off between fidelity of the control frequency and computational burden.

5.3. Reward model

The reward function is one of the main characters, if not the main one, when talking about Reinforcement Learning. It drives the agent policy, that aims at maximizing it, by means of positive and negative scores, which should incentivize a specific agent behavior. It should phrase the objectives and constraints of the problem in mathematical form. For this work, several scores have been defined to create the reward model; different selections of scores can be used to design different reward models, depending on the case or training considered. The following list collects all the scores the authors used to defined the different reward models.

- *distance score*: in the case of proximity operations, a general and intuitive idea is that the chaser spacecraft shall not crash onto the target, nor escape far away from it. This constraint is formulated adopting a lower and upper limit in terms of relative distance between the two objects (specifically between their center of masses). In this way it is intrinsically introduced safety in the operations, by incentivizing the agent to avoid dangerous regions of space, in which the mission would completely fail. The associated mathematical expression and score are reported in Eq. 12:

$$r_d = \begin{cases} -100 & \text{if } d \leq D_{min} \text{ or } d \geq D_{max} \\ 1 & \text{otherwise} \end{cases} \tag{12}$$

  where d in this case indicates the norm of the distance vector between chaser and target center of masses, with $D_{min} = 50m$ and $D_{max} = 500m$.

- *incidence angle score*: regarding the main goal of the presented work, the agent should maximize a reward function that enables it to better map the target. This request is connected to the adopted mapping technique, that would require to assert some specific conditions in terms of incidence angle and number of quality images per mesh face, as discussed in Sec. 2. At each timestep the agent keeps track of the target rotation and re-computes the normal direction for each of the mesh faces. First, a screening of the faces that are in the field of view of the spacecraft's cameras is performed. Then, the angle between each of the normal directions of the faces in visibility and the camera vector, assumed as continuously pointing the target center, is calculated. A score is formulated on this resulting incidence angle ε and reported in Eq. 13:

$$r_\varepsilon = \begin{cases} 1 & \text{if } 10^\circ \leq \varepsilon \leq 50^\circ \\ \frac{1}{5}\varepsilon - 1 & \text{if } 5^\circ \leq \varepsilon \leq 10^\circ \\ 6 - \frac{1}{10}\varepsilon & \text{if } 50^\circ \leq \varepsilon \leq 60^\circ \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

- *emission angle score*: the Sun incidence angle η is the angle between the Sun direction and the normal vectors to the target mesh faces. The same considerations made for the incidence angle, regarding the relation to the mapping technique and the computation of η, are still valid. This angle should be between $20^\circ - 60^\circ$, to avoid shadows or excessive brightness, that may affect the good quality of the image. Some margin is added, as expressed in Eq. 14.

$$r_\eta = \begin{cases} 1 & \text{if } 20^\circ \leq \eta \leq 60^\circ \\ \frac{1}{10}\eta - 1 & \text{if } 10^\circ \leq \eta \leq 20^\circ \\ 7 - \frac{1}{10}\eta & \text{if } 60^\circ \leq \eta \leq 70^\circ \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

- *map percentage score*: to better reconstruct the target geometry and shape, a reward on the current level of the map is necessary. The overall map is fragmented into a number N_p of quality photos for each face constituting the mesh, where quality is to be intended with respect to the incidence angles ε and, depending on the case, emission angle η between the camera and the face. At each time step, the map percentage can be computed counting the number of quality pictures ($r_\varepsilon \neq 0$ and $r_\eta \neq 0$) available for each face N_q up to that moment and dividing this quantity by N_p times the number of mesh faces n_{faces}, as in Eq. 15. Quality pictures are to be intended in terms of the incidence and emissivity angles defined before. At each time step, the algorithm checks which faces of the mesh are in visibility of the camera, and a picture of one of these faces is said to be of "good quality" if the reward signals r_ε and r_η associated to that single face are greater than zero.

$$M_{\%,k} = \frac{N_q}{N_p * n_{faces}} \tag{15}$$

$$r_m = \begin{cases} 1 & \text{if } M_{\%,k} > M_{\%,k-1} \\ 100 & \text{if } M_{\%,k} = 100 \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

In Eq. 16, note how the agent is rewarded for improving the map level and it is also given a big bonus for completing the map reconstruction.

Once all the mathematical expressions, defining the problem objectives, are detailed, the overall reward function is simply the sum of these terms. Different reward models have been used for the training and testing phases. For example, in terms of vision-based navigation and imaging system. two different models have been established, depending on the usage of only a visible camera (VIS) or the sensor fusion between a visible and thermal cameras (VIS+TIR). The former is referred to as R_{VIS}, while the second as $R_{VIS,TIR}$.

$$R_{VIS} = r_d + r_\varepsilon + r_\eta + r_m \tag{17}$$

$$R_{VIS,TIR} = r_d + r_\varepsilon + r_m \tag{18}$$

In the second expression the reward regarding the emission incidence angle is neglected, since the vision-based architecture assumes the presence of also thermal infrared imaging, thus nullifying the problems of shadowing and poor illumination.

6. Neural Network models comparison

Neural Networks (NN) are a powerful tool for function approximation and, as such, they become attractive in the DRL context to simulate the agent policy. NNs have indeed the ability to learn and model non-linear and complex relationships, and to generalize the results, meaning that they can infer input-output mappings on unseen data. These key advantages make them a solid and robust candidate when in need to approximate a certain behaviour, and depending on they architecture they gain specific characteristics.

One of the main goal of this work is the performance comparison between two agents defined by two different neural network models: a simple and classic multi-layers feed-forward neural network architecture, already developed in (Brandonisio, 2019-2020), and a recurrent neural network architecture. For a complete overview of this analysis please refer to (Brandonisio & Lavagna, 2021). An illustrative comparison between the two networks models can be inferred looking at Fig. 2 and Fig. 3.

Feed-forward neural networks (FFNN) allow signals to travel one way only: from input to output. There are no feedback (loops); i.e., the output of any layer does not affect that same layer. The most widely used and studied FFNN is the Multi-Layer Perceptron (MLP), which is also the one employed in this work. The simple mathematical expression relating the input to the output between two adjacent layers is reported in Eq. 19.

$$\mathbf{q}^{i+1} = \sigma\left(\mathbf{W}^i\mathbf{q}^i + \mathbf{b}^i\right) \tag{19}$$

where $\mathbf{q}^{i+1}$ represents the vector of activation values for the neurons in layer $i+1$, σ is the activation function, $\mathbf{W}^i$ is the matrix containing all the weights connecting neurons in layer i and $i+1$, $\mathbf{q}^i$ is the vector of activation values for the neurons in layer i, and $\mathbf{b}^i$ is the vector of biases for neurons in layer i. For a more detailed discussion about MLP refer to Goodfellow et al. (2016).

Differently from FFNN, recurrent neural networks (RNN) introduce loops: computations derived from earlier inputs are fed back into the network, and then fed forward to be processed into outputs. Thus, they could take advantage of time correlation in the data and be more stable. Among the different types of RNN, in this work, the Long Short-Term Memory (LSTM) recurrent layer is exploited. For each input vector the recurrent layer perform the following computations:

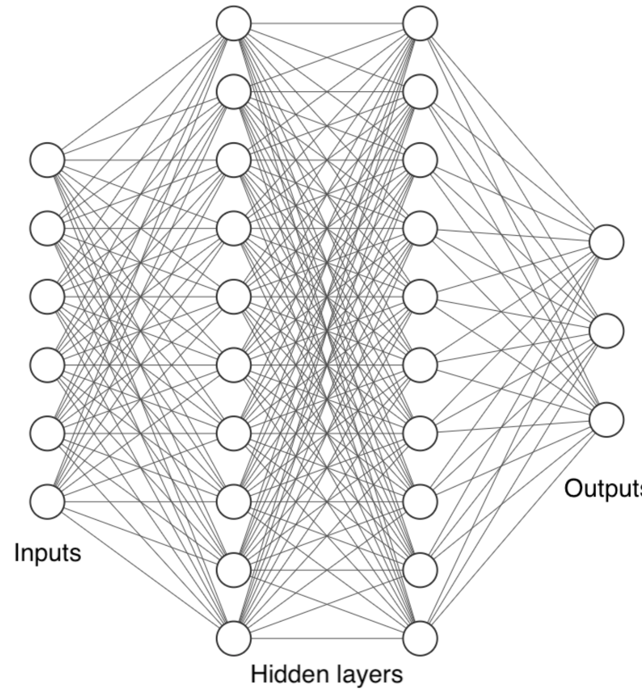


Fig. 2: Graphic representation of the FFNN architecture.

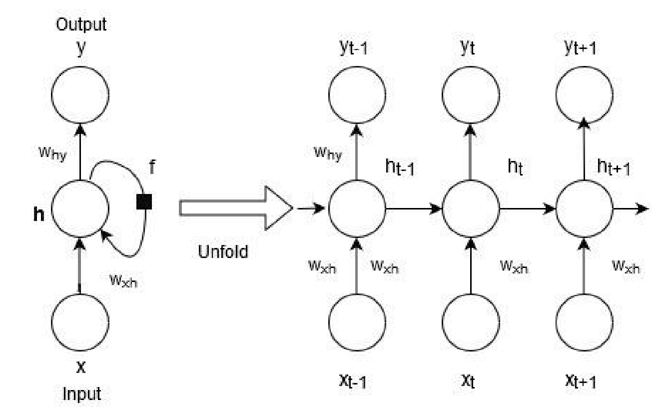


Fig. 3: Graphic representation of the RNN architecture (Paramasivan, 2021).

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^i\mathbf{x}_t + \mathbf{b}^i + \mathbf{W}^i_h\mathbf{h}_{t-1} + \mathbf{b}^i_h) \\
\mathbf{f}_t &= \sigma(\mathbf{W}^f\mathbf{x}_t + \mathbf{b}^f + \mathbf{W}^f_h\mathbf{h}_{t-1} + \mathbf{b}^f_h) \\
\mathbf{g}_t &= \tanh(\mathbf{W}^g\mathbf{x}_t + \mathbf{b}^g + \mathbf{W}^g_h\mathbf{h}_{t-1} + \mathbf{b}^g_h) \\
\mathbf{o}_t &= \sigma(\mathbf{W}^o\mathbf{x}_t + \mathbf{b}^o + \mathbf{W}^o_h\mathbf{h}_{t-1} + \mathbf{b}^o_h) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{q}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{20}$$

where $\mathbf{h}_t$, $\mathbf{c}_t$ and $\mathbf{x}_t$ are the hidden state, the cell state and the input at time t, $\mathbf{h}_{t-1}$ is the hidden state of the layer at time *t-1*; $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{g}_t$ and $\mathbf{o}_t$ are the input, forget, cell and output gates respectively. σ is the sigmoid activation function and $\odot$ is the Hadamard product. The overall process followed in Eq 20 can be visualized in Fig. 4, where a single LSTM cell schematic is shown. For a more detailed explanation about LSTM refer to Sak et al. (2014).

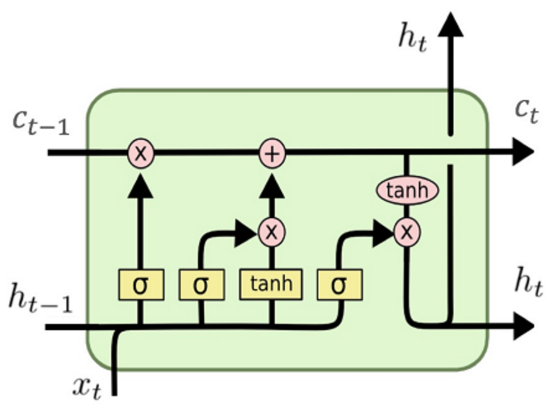


Fig. 4: LSTM scheme. *Source:* `https://commons.wikimedia.org/wiki/File:LSTM.png`

The idea behind the formulation of the PPO with recurrent neural network is related to the potential benefits that such an architecture can have in terms of training and performance. Indeed, recurrent networks have the capability to store past states information, thus it may strongly affect the agent safe trajectories planning to faster achieve the mission goals. In addition, training an RNN may be beneficial to refine the agent's environmental conditions sensitivity, increasing its robustness, regardless the specific operational environment. Therefore, we considered important to compare both architectures to understand if an improvement of the stability and sensitivity can be possible with respect to the particular conditions in which the problem is solved. In the next subsections the two models will be described and afterwards the results will be presented. For this particular analysis, the *discrete* action space model is used, with a reward model based only on a visible camera (R_{VIS}). The target object, in this case, is shaped as a simple rectangular parallelepiped.

In this PPO implementation, both the policy and the value functions (actor and critic networks) are learned concurrently. The action space models is *discrete*, thus implies the use of a softmax activation function to select the action at each time step, among the different possible options, previously described in Sec. 5.2. The output of the softmax activation function is a multi-categorical distribution, among which the policy samples the action to take during the optimization process. The main parameters related to the loss functions are the reward discount factor γ, the terminal reward discount factor λ, and the clipping parameter ϵ. The first is the factor that multiplies the reward at each time step of a simulation episode, as defined in Sec.3.1; it is set to 0.99. Instead, the second parameter λ, is the factor the multiplies the overall sum of rewards at the end of a single episode simulation and it is set to 0.94. The clipping factor, defined in Sec. 4, is 0.2. The optimization periodically updates the policy and value functions with information collected during 10 episodes trajectories. Afterwords, the data are divided in batch of dimension 32, and used for 5 epochs updates. The terminal conditions for an episode are the complete acquisition of the target object map, the spacecraft escape from the region defined by the minimum and maximum distance from the target and lastly the exceeding of the time window, even if this last option is very unlikely to occur.

6.1. Feed-forward Neural Network Architecture

The feed-forward neural network architecture for the policy and value functions are equally defined. They are composed by three linear layers with tanh and Leaky-ReLU as activation functions. The architecture is described in Table 1 and Table 2, where *dim_obs* is the observation state dimension (corresponding to the state vector dimension, defined in Sec. 5.1), *dim_act* is the action space dimension and n_{h1}, n_{h3} are the first and third hidden layer dimensions respectively. In order to improve the convergence and avoid saturation problem the tanh-layers are initialized as semi-orthogonal matrices, as suggested by Saxe et al. (2014).

	Policy Network	
Layer	Elements	Activation
1^{st} Hidden Layer (h1)	10*dim_obs	tanh
2^{nd} Hidden Layer (h2)	$\sqrt{n_{h1} * n_{h3}}$	tanh
3^{rd} Hidden Layer (h3)	10*dim_act	Leaky-ReLU
output	dim_act	softmax
Learning rate	10^{-5}	

Table 1: Policy Network Architecture: Linear Case

	Value Network	
Layer	Elements	Activation
1^{st} Hidden Layer (h1)	10*dim_obs	tanh
2^{nd} Hidden Layer (h2)	$\sqrt{n_{h1} * n_{h3}}$	tanh
3^{rd} Hidden Layer (h3)	10*dim_act	Leaky-ReLU
output	dim_act	linear
Learning rate	10^{-5}	

Table 2: Value Network Architecture: Linear Case

6.2. Recurrent Neural Network Architecture

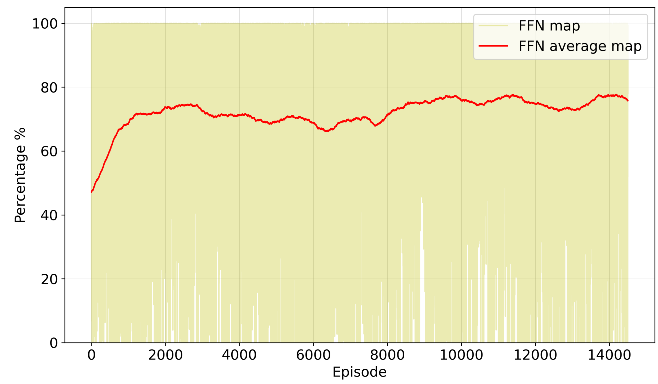
In the recurrent network case, the architecture is defined coupling one LSTM recurrent layer and two drop-out linear layers (Sak et al., 2014). The models for the policy and value networks are shown in Table 3 and Table 4. Also here the activation functions are equally selected for both networks.

	Policy Network	
Layer	Elements	Activation
LSTM Layer	24	-
1^{st} Hidden Layer (h1)	64	ReLU
2^{nd} Hidden Layer (h2)	32	ReLU
output	dim_act	softmax
Learning rate	10^{-5}	

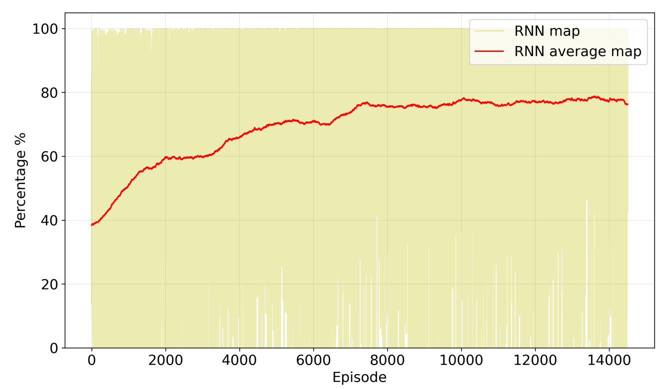
Table 3: Policy Network Architecture: Recurrent Case

	Value Network	
Layer	Elements	Activation
LSTM Layer	24	-
1^{st} Hidden Layer (h1)	64	ReLU
2^{nd} Hidden Layer (h2)	32	ReLU
output	dim_act	linear
Learning rate	10^{-5}	

Table 4: Value Network Architecture: Recurrent Case



(a) Feed-forward Network



(b) Recurrent Network

Fig. 5: Comparison between feed-forward and recurrent architectures in the simplest random conditions case.

6.3. Results

In order to bound the problem some characteristics have been maintained constant during the overall training procedure. In particular the camera field of view (FOV) is fixed as 10°, that can be considered as a common FOV for space optical cameras; the integration time is fixed at 30s and the accuracy level N_p for the map is fixed at 25 correct photos per face. The scenario initial conditions are random in terms of Sun initial phase, target object orbital true anomaly and rotational dynamics (angle position and velocity). In Figure 5, the training of the two models is shown. In the plot the average map level trends of the feed-forward and recurrent policy architectures are compared in a training simulation of 15000 episodes length.

Some important remarks can be derived from the presented results:

- The linear policy seems to learn and converge faster then the recurrent policy. Nevertheless, the linear-case curve presents more oscillations and an overall lower stability with respect to the recurrent curve policy behaviour.

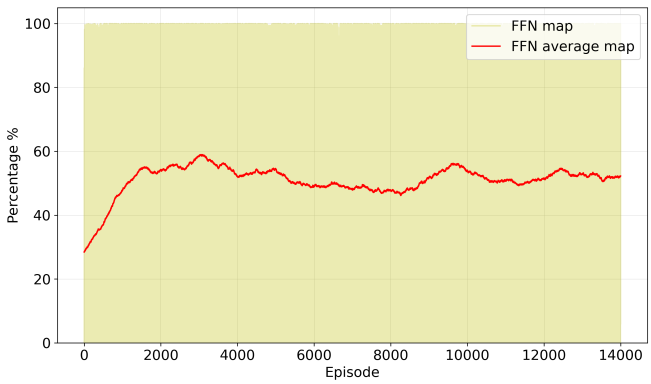
- Concerning the final result of the simulation, the recurrent policy converges to a slightly higher map level in the same training length of the linear policy. On average the map level reached by the two policies is around 70%-80%.

- The fact that the learning curve of the recurrent policy grows gradually confirms, as expected, that the learning process is slower but safer and more stable. Indeed, the potential robustness of a recurrent network was one of the main reason that drove this kind of analysis.

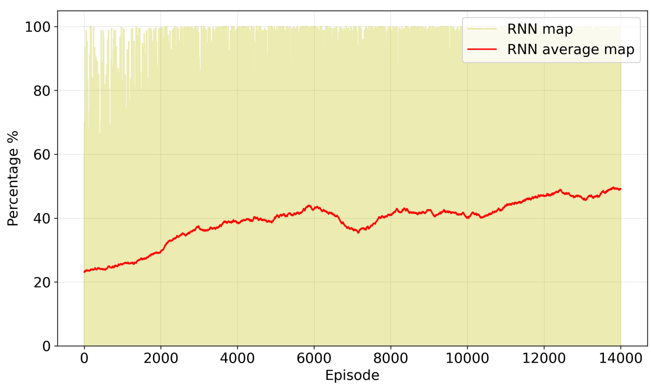
The two models have been trained also with an higher randomness level adding random conditions for chaser-target initial relative position. Their relative position is however constrained to have both the x, y and z coordinates positive. The rational behind this assumption derives from the will of containing the complexity of the problem in order not to saturate the neural network learning capabilities and also simulate a possible real scenario, in which the initial condition is constrained in a specific space without knowing a priori the correct engagement position. In Figure 6, the results obtained are shown. In the plot the average trends of the feed-forward and recurrent policy architectures are compared again for 15000 episodes length simulation. The level reached by the two policies is comparable and it settles around 55%. Also here, as in the previously analysis, the characteristics of the different architectures hold. Considering the fact that the state space is much bigger now, due to the randomness in the initial relative position, the average map level is lower than the one achieved in the previous case for the same amount of training episodes, as expected.

7. Continuous Action Space agent

In this section, the transition from *discrete action space* to *continuous* is discussed (Capra, 2020-2021). The major dif-



(a) Feed-forward Network



(b) Recurrent Network

Fig. 6: Comparison between feed-forward and recurrent architectures in the most complex random conditions case.

ferences between the two have already been highlighted in Sec. 5.2, and now a PPO agent working in continuous action space is developed, in a slightly different scenario with respect to the previous one. This type of control workspace is much more realistic with respect to a discrete one, but at the same time it is much more computationally expensive, due to the high dimensionality that the policy needs to analyze.

A feed-forward network, similar to the one in Sec 6.1, is implemented and the architecture adopted is reported in Table 5, for both the actor and the critic networks.

For this analysis the reward function considers a vision-based system employing multi-spectral cameras, so both visible and thermal infrared. Therefore, all the results are referred to the $R_{VIS,TIR}$ reward expression. Notably this objective function is simpler, but this selection is justified by the intrinsic complexity of having a continuous action space, which would require a much longer training.

Moreover, a different target is considered, replacing the artificial rectangular parallelepiped with a triangular mesh of VESPA (Vega Secondary Payload Adapter), which is a space object orbiting the Earth as a debris, and it is gaining much interest by space agencies, targeting it for future missions (Silvestrini et al., 2021). The simulation conditions, as well as the PPO parameters are the same as of the previous discrete analysis in Sec 6, except for the batch size, as a larger value of 512 for its dimension is found to be more suitable in the case of a continuous action space.

	Actor	Critic
Layer	Neurons	Neurons
Input	dim_obs	dim_obs
1^{st} hidden layer	256	256
2^{nd} hidden layer	256	256
Output	3	1
Learning rate	10^{-5}	5×10^{-5}
Activation function	Tanh	Tanh

Table 5: Actor & Critic Network specifications with a continuous action space.

The agent training is performed on $N_{episodes} = 30000$, which is exactly double the episodes for the discrete action space scenario. Once again this is justified by the much higher dimensionality of the problem, and by the generality of the initial conditions, which are generated randomly for both the relative position and target attitude, as in Table 6.

Variable	Range
d	$2D_{min} < d < 0.5D_{max}$
α	$0° < \alpha < 360°$
δ	$-90° < \delta < 90°$
v	$0\,m/s$
θ_i	$0°$
$\dot{\theta}_i$	$-0.001rad/s < \dot{\theta}_1 < 0.001rad/s$

Table 6: State variables initial condition ranges.

d and v are the relative position and velocity between the chaser and the target, α and δ represents the azimuth and the elevation angle respectively, and finally θ and $\dot{\theta}$ expresses the rotation angles and velocity of the target, with $i \in [1:3]$ specifying the axis. D_{min} and D_{max} are the two boundaries defined in Sec. 5.3.

The average map level profile during the training is reported in Fig. 7.

Some notable remarks are critically commented next:

- the performance level is good, peaking at about 95% of covered map, so the training step can be considered successful;

- the profile of the average map increases over the span of the episodes, and seems to be still improving, suggesting that a longer training could be beneficial.

An example of trajectory, completing 100% of the map, is shown in Fig. 8.

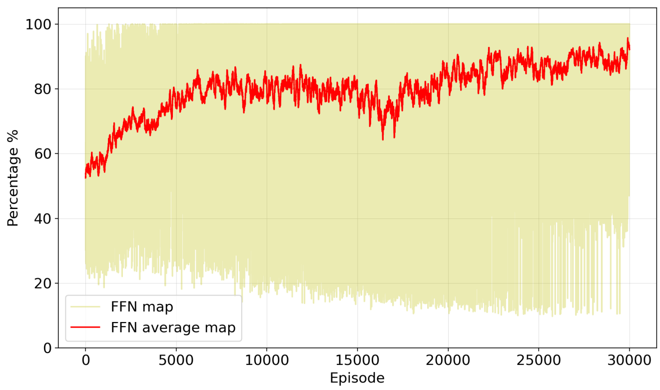


Fig. 7: Average map percentage covered during the agent training.

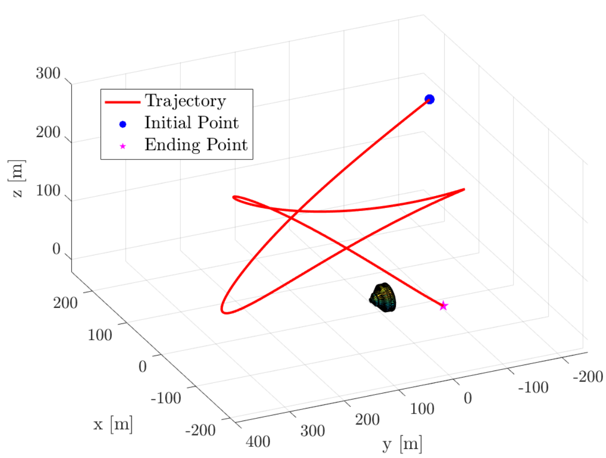


Fig. 8: Example trajectory.

7.1. Benchmark testing

This section reports the tests carried out to assess the performance of the model discussed up to now, against some simple benchmarks, to check the effectiveness of the learning step and of the reward function design. The first two comparisons are against *no-learning* models, which essentially means that they have not gone through the training procedure:

- the first simply propagates the free-dynamics from the random i.c.;
- the second is a random control model.

The average map percentage obtained by both models, starting from random initial conditions, is reported in Table 7.

Free-Dynamics	Random Control
52.9%	55.5%

Table 7: Baseline models map percentage.

The principal model performs much better than both of them, verifying the training effectiveness.

A further benchmark test is performed by comparing the performance with a model that undergoes the training step, but with a simpler reward function, entailing just the chaser-target distance objective. As such, the agent learns how to remain in proximity of the target, keeping itself in the safe region of space, but it does not learn how to map the it efficiently, because no information regarding the map level and the quality of images is fed to its policy network. This will be referred to as the ”simple” model, to differentiate it from the principal one. The simple model performs worse than the principal one (reaching about 73% of average map level), confirming the good design of the reward function, which incentivizes the agent to better perform the shape reconstruction. Moreover, this new agent takes much longer, on average, to complete the map, as it can be seen in Table 8, since its main objective is to simply remain inside the boundaries in space.

	Principal model	Simple model
$t_{100\%}[s]$	1595	3150

Table 8: Average time to complete the map.

The principal model takes nearly half the time to cover 100% of the target map, thus confirming that it has learnt a different, more efficient strategy for mapping VESPA, than simply remaining inside the limits.

7.2. Discrete vs Continuous Action Space comparison

As both the discrete and the continuous action space PPO agents have been designed, the aim of this section is to compare the results obtained by these two models. To keep the comparison consistent, they have to be applied in the same scenario, which for the scope of this analysis is the one presented in Sec 7. The two models are trained for the same amount of episodes, and the result of the discrete action space agent is reported in Fig 9. Note that the same result for the continuous action space agent is the one discussed before in Fig 7.

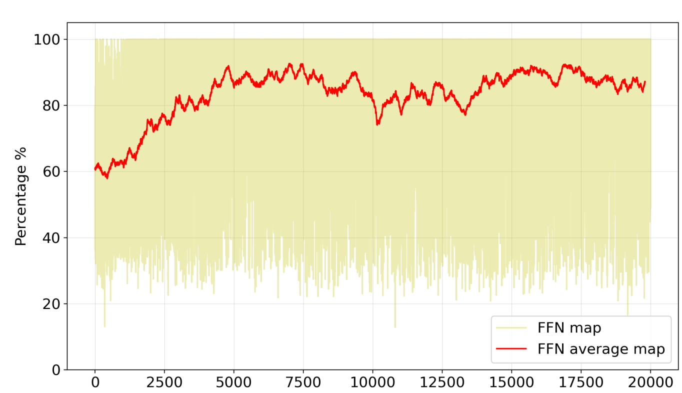


Fig. 9: Average map percentage profile during discrete action agent training.

The output map level profiles are now commented in details:

- the discrete agent reaches training convergence much faster, peaking after a few thousands episodes. This is due to the much simpler and exceedingly smaller action space in the discrete case, since the agent can select between just 7 thrust control impulses, as in Eq. 11, instead of practically infinite possibilities as it happens in a continuous action space. Thus, the training is much faster and requires fewer episodes;

- in terms of performance level, it gets to about 90% of average map level, which, apart from the fact that the scenario is different, is much higher than the previous test in Sec. 6. This is due to the removal of the reward constraint on the emission angle, linked to the illumination conditions. In this scenario it is easier to obtain quality images of the target mesh faces;

- peaking at 90% of average map level, it falls just a little short of the continuous action space model. This could be imputed to the greater flexibility guaranteed by a continuous action space.

This comparison sets an important step towards a more realistic and refined control of the spacecraft motion with Deep Reinforcement Learning, at the expenses of a longer and computationally heavier training.

8. Robustness & Sensitivity analysis

In this section the performance of the continuous action space agent developed in Sec. 7 are evaluated against previous unseen scenarios, verging on the following aspects:

- swap the linearized eccentric dynamics used during training with more complex nonlinear models, that should represent with more fidelity the real evolution of the relative motion between the chaser and the target;

- introduce random noise in the relative motion estimation. The pose is retrieved from navigation with the sensors onboard (in this case vision-based), which are affected by errors in their measurements;

- a sensitivity analysis on the rotational velocity of the target is carried out, by investigating the effects of a faster attitude motion.

8.1. Nonlinear dynamics

Two nonlinear relative dynamics models are considered: unperturbed (Sullivan et al., 2017) and J_2 perturbed (Xu & Wang, 2008).

The difference between these two models and the linearized eccentric employed during training, can be appreciated in Fig. 10, where the free dynamics is propagated from the same initial conditions.

Note that the difference between the models is quite negligible, thus suggesting that the agent should be capable of performing well also when the dynamics is not the one it was trained on. This logic assumption is supported by the results in terms of average map obtained running the simulation tests and reported in Table. 9.

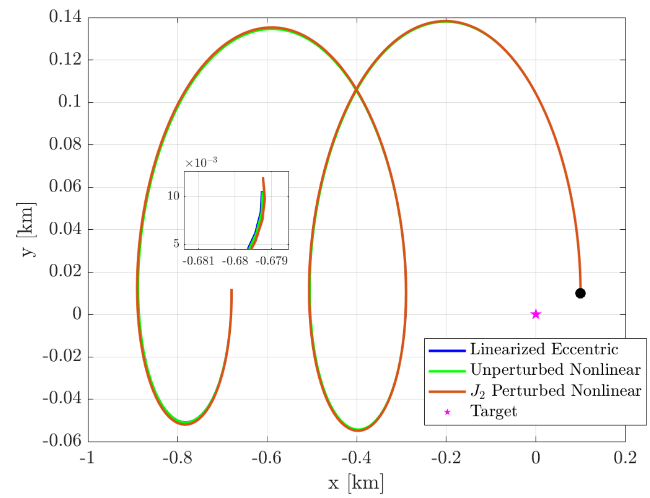


Fig. 10: Comparison between linearized eccentric and non linear free dynamics.

	Map [%]
Linearized Eccentric	95.12%
Unperturbed Nonlinear	94.37%
J_2 Perturbed Nonlinear	94.57%

Table 9: Map percentage comparison between different models.

8.2. Navigation uncertainty

During the training and testing, the state variables were assumed to be correct and perfectly known. However, in a more realistic scenario, uncertainty is strongly present due to the errors, for as small they are, in sensors measurements. Moreover, modeling errors are always present and affect the truthfulness of computer simulations. The aim of this section is to investigate what happens to the performance if noise is added at each time-step between the estimated value coming from navigation and the guidance block. Specifically, noise is added to the relative position and velocity vector components, sampling from Gaussian distributions defined by the following standard deviations:

$$\sigma_{pos} = 10\,m \qquad \sigma_{vel} = 0.1\,m/s$$

The first test is a simulation in which noise is applied to both variables and the performance level experience a reduction, reaching about 80% of average map level.

The same test is performed applying distinctively the two uncertainties, first on the relative position and then on the velocity. Table. 10 summarizes the results of all tests.

The model is not so robust in this scenario, so a deeper analysis is deemed necessary to better understand how uncertainty affects the performance level.

Position + Velocity	Position	Velocity
80.38%	84.07%	87.58%

Table 10: Map percentage comparison with navigation uncertainty.

8.3. *Target attitude analysis*

The range for the starting target attitude motion was selected looking at the values in Table. 6, which defined the initial conditions. In this section a sensitivity analysis on the angular velocity is carried out, by comparing the results of two simulations: fast target attitude and slow target attitude. The case of slow attitude is the one studied up to now, while for the fast case, the range, from which the initial attitude motion gets sampled, is enlarged:

$$|\dot{\theta}_i| < 0.001\, rad/s \quad \longrightarrow \quad |\dot{\theta}_i| < 0.005\, rad/s$$

The principal model is then tested with this modification and the performance level falls off a cliff with respect to the results in the nominal training case, as the agent can only cover 69% of the map on average. There are two main reasons associated to this result:

- the state space is greatly augmented;
- the agent policy network adjusted its parameters heavily influenced by the target rotational velocity. This seems to be intuitive, since the agent selects its next actions depending on how VESPA is rotating, to plan a trajectory that can inspect the faces it has yet to see.

To solve for this issue, a new model training is set-up, keeping the same architecture and parameters used before and simply enlarging the rotational velocity range to the one employed during the test.

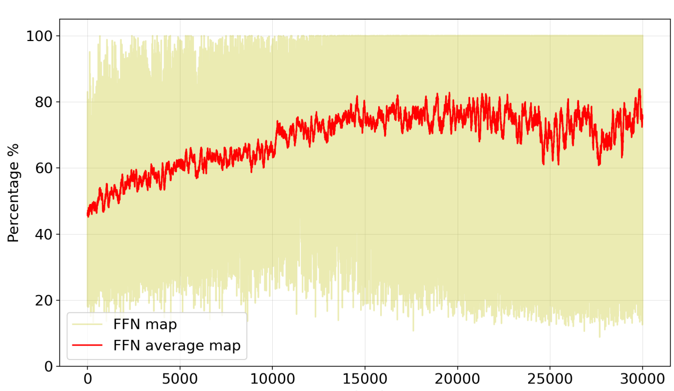


Fig. 11: Average map percentage during fast attitude training.

Notable remarks are now discussed:

- the agent improves its performance level, reaching about 80% on average of map reconstructed, as visible in Fig. 11;
- the performance level is lower than the one obtained for the principal model, but this is expected. Indeed, the state space has been greatly expanded, so a training procedure with the same number of episodes will inevitably bring to worst results;
- the agent is still capable of learning a quality policy and if trained for a higher number of episodes, the result would be most probably even better.

Therefore, a faster target rotational dynamics does not seem to be a bottleneck for the model performance, but rather this extension of the state space simply makes the required training step longer.

9. Conclusion

This work presented an in depth analysis of an innovative autonomous guidance algorithm for the shape reconstruction of an uncooperative space object, developed via Deep Reinforcement Learning.

Starting from the reference point set by Brandonisio (2019-2020), the method has been refined, by comparing the performance obtained with different neural network architectures, in the case of a discrete action space. Specifically, RNNs improves training stability and reduces oscillations, although with respect to simple MLPs, the end result is almost the same.

A further important step forward in the investigation of Proximal Policy Optimization for solving the spacecraft decision-making policy is made, by implementing it with a continuous action space, to simulate more realistically the actuator control on the chaser motion. The two models, *discrete* and *continuous* action space, are then compared in the same scenario, and the result is critically commented. The continuous action space model is then extensively tested to asses its performance, robustness and sensitivity against unseen conditions.

As a result, the work confirmed and developed further the applicability of DRL algorithms to the spacecraft autonomous guidance problem, applied for the shape reconstruction of an uncooperative target.

9.1. *Future developments*

Starting from the results obtained by this work, further improvements can be made and a few possibilities are reported in the following.

Modeling of the chaser attitude dynamics would bring several benefits:

- make the model more representative of real conditions;
- the agent could also select autonomously when it is the right moment to perform a slew maneuver, or switch the cameras on, depending on if the target is in the field of view or not;
- elongated objects could be considered since the cameras are no more restricted to point towards the target center of mass.

Further developments could be made augmenting the reward function, by introducing an expression to incentivize a faster map reconstruction, lower propellant consumption (Brandonisio & Lavagna, 2021), or new tasks that the spacecraft should perform.

Uncertainties in the pose estimation, as well as intrinsic error due to the selected models need to be analyzed in greater details. Concerning this aspect, a Model-Based Reinforcement Learning (MBRL) framework could be set-up to help guiding the agent and decrease sensitivity to noise in the measurement by online learning the underline dynamics.

Finally, in systems with limited hardware resources, like a small spacecraft, effective pruning and shrinking techniques might be a solution to the problem of high computational cost and memory consumption, that are limiting the applicability of Deep Reinforcement Learning algorithms.

References

Brandonisio, A. (2019-2020). *Deep Reinforcement Learning to Enhance Fly-around Guidance for Uncooperative Space Objects Smart Imaging*. Master's thesis Politecnico di Milano.

Brandonisio, A., & Lavagna, M. (2021). Sensitivity analysis of adaptive guidance via deep reinforcement learning for uncooperative space objects imaging. In *2021 AAS/AIAA Astrodynamics Specialist Conference* (pp. 1–20).

Brandonisio, A., Lavagna, M., & Guzzetti, D. (2021). Reinforcement learning for uncooperative space objects smart imaging path-planning. *The Journal of the Astronautical Sciences*, *68*(4), 1145–1169. doi:10.1007/s40295-021-00288-7.

Capra, L. (2020-2021). *Deep Reinforcement Learning towards adaptive Vision-Based autonomous Guidance*. Master's thesis Politecnico di Milano.

Chan, D. M., & Agha-mohammadi, A.-a. (2019). Autonomous imaging and mapping of small bodies using deep reinforcement learning. In *2019 IEEE Aerospace Conference* (pp. 1–12). doi:10.1109/AERO.2019.8742147.

Civardi, G. L., Piccinin, M., & Lavagna, M. (2021). Small bodies ir imaging for relative navigation and mapping enhancement. In *7th IAA Planetary Defense Conference*.

Downes, L. M., Steiner, T. J., & How, J. P. (2020). Lunar terrain relative navigation using a convolutional neural network for visual crater detection. In *2020 American Control Conference (ACC)* (pp. 4448–4453). doi:10.23919/ACC45564.2020.9147595.

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, *13*(2), 99–110. doi:10.1109/MRA.2006.1638022.

Emami, E., Ahmad, T., Bebis, G. et al. (2019). Crater detection using unsupervised algorithms and convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, *57*(8), 5373–5383. doi:10.1109/TGRS.2019.2899122.

Furfaro, R., Bloise, I., Orlandelli, M. et al. (2018). Deep learning for autonomous lunar landing. In *2018 AAS/AIAA Astrodynamics Specialist Conference* (pp. 3285–3306). Univelt volume 167.

Gaskell, R. W. (2001). Automated landmark identification for spacecraft navigation. *Advances in the Astronautical Sciences*, *109*, 1749–1756.

Gaudet, B., Linares, R., & Furfaro, R. (2020a). Deep reinforcement learning for six degree-of-freedom planetary landing. *Advances in Space Research*, *65*(7), 1723–1741. doi:https://doi.org/10.1016/j.asr.2019.12.030.

Gaudet, B., Linares, R., & Furfaro, R. (2020b). Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations. *Acta Astronautica*, *171*, 1–13. doi:https://doi.org/10.1016/j.actaastro.2020.02.036.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hovell, K., & Ulrich, S. (2021). Deep reinforcement learning for spacecraft proximity operations guidance. *Journal of Spacecraft and Rockets*, *58*(2), 254–264. doi:10.2514/1.A34838.

Inalhan, G., Tillerson, M., & How, J. P. (2002). Relative dynamics and control of spacecraft formations in eccentric orbits. *Journal of Guidance, Control, and Dynamics*, *25*(1), 48–59. doi:10.2514/2.4874.

Kurniawati, H. (2021). Partially observable markov decision processes (pomdps) and robotics. *CoRR*, *abs/2107.07599*. URL: https://arxiv.org/abs/2107.07599.

Martínez, J., Rafalskyi, D., & Aanesland, A. (2019). Development and testing of the npt30-i2 iodine ion thruster. In *36th International Electric Propulsion Conference*. doi:10.6084/m9.figshare.11931363.

Mnih, V., Badia, A. P., Mirza, M. et al. (2016). Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, (pp. 1928–1937). arXiv:1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. doi:https://doi.org/10.1038/nature14236.

Paramasivan, S. (2021). Deep learning based recurrent neural networks to enhance the performance of wind energy forecasting: A review. *Revue d'Intelligence Artificielle*, *35*(1), 1–10. doi:https://doi.org/10.18280/ria.350101.

Pesce, V., Agha-mohammadi, A.-a., & Lavagna, M. (2018). Autonomous navigation and mapping of small bodies. In *2018 IEEE Aerospace Conference* (pp. 1–10). doi:10.1109/AERO.2018.8396797.

Piccinin, M., Lunghi, P., & Lavagna, M. (2022). Deep reinforcement learning-based policy for autonomous imaging planning of small celestial bodies mapping. *Aerospace Science and Technology*, *120*, 107224. doi:https://doi.org/10.1016/j.ast.2021.107224.

Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. arXiv:1402.1128.

Saxe, A. M., McClelland, J. L., & Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *International Conference on Learning Representations*, . arXiv:1312.6120.

Schulman, J., Levine, S., Abbeel, P. et al. (2015). Trust region policy optimization. In F. Bach, & D. Blei (Eds.), *Proceedings of the 32nd International Conference on Machine Learning* (pp. 1889–1897). Lille, France: PMLR volume 37 of *Proceedings of Machine Learning Research*. URL: https://proceedings.mlr.press/v37/schulman15.html.

Schulman, J., Wolski, F., Dhariwal, P. et al. (2017). Proximal policy optimization algorithms, . arXiv:1707.06347.

Silvestrini, S., & Lavagna, M. (2021a). Neural-aided gnc reconfiguration algorithm for distributed space system: development and pil test. *Advances in Space Research*, *67*(5), 1490–1505. doi:https://doi.org/10.1016/j.asr.2020.12.014.

Silvestrini, S., & Lavagna, M. (2021b). Neural-based predictive control for safe autonomous spacecraft relative maneuvers. *Journal of Guidance, Control, and Dynamics*, *44*(12), 2303–2310. doi:https://doi.org/10.2514/1.G005481.

Silvestrini, S., Piccinin, M., Zanotti, G. et al. (2022). Optical navigation for lunar landing based on convolutional neural network crater detector. *Aerospace Science and Technology*, *123*, 107503. doi:https://doi.org/10.1016/j.ast.2022.107503.

Silvestrini, S., Prinetto, J., Zanotti, G. et al. (2021). Design of robust passively safe relative trajectories for uncooperative debris imaging in preparation to removal. In *Advances in the Astronautical Sciences* (p. 4205 – 4222). volume 175. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85126240899&partnerID=40&md5=6a51911e8e10ed060ac72ea48b7bbcb5 cited by: 0.

Sullivan, C. J., & Bosanac, N. (2020). Using reinforcement learning to design a low-thrust approach into a periodic orbit in a multi-body system. In *AIAA Scitech 2020 Forum*. doi:10.2514/6.2020-1914.

Sullivan, J., Grimberg, S., & D'Amico, S. (2017). Comprehensive survey and assessment of spacecraft relative motion dynamics models. *Journal of Guidance, Control, and Dynamics*, *40*(8), 1837–1859. doi:10.2514/1.G002309.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tatsch, A., Fitz-Coy, N., & Gladun, S. (2006). On-orbit servicing: A brief survey. In *Proceedings of the IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR'06)* (pp. 276–281).

Xu, G., & Wang, D. (2008). Nonlinear dynamic equations of satellite relative motion around an oblate earth. *Journal of Guidance, Control and Dynamics*, *31*(5), 1521–1524. doi:https://doi.org/10.2514/1.33616.