Collision Avoidance and Return Manoeuvre Optimisation for Low-Thrust Satellites Using Reinforcement Learning

Alexandru Solomon and Ciprian Paduraru
University of Bucharest, Department of Computer Science, Bucharest, Romania

Keywords: Collision Avoidance, Return Manoeuvre, Manoeuvre Optimisation, Reinforcement Learning, DQN, PPO, REINFORCE.

Abstract: Collision avoidance is an essential aspect of day-to-day satellite operations, enabling operators to carry out their missions safely despite the rapidly growing amount of space debris. This paper presents the capabilities of reinforcement learning (RL) approaches to train an agent capable of collision avoidance manoeuvres for low-thrust satellites in low-Earth orbit. The collision avoidance process performed by the agent consists of optimizing a collision avoidance manoeuvre as well as the return manoeuvre to the original orbit. The focus is on satellites with low thrust propulsion systems, since the optimization process of a manoeuvre performed by such a system is more complex than for an impulsive system and therefore more interesting to be solved by RL methods. The training process is performed in a simulated environment of space conditions for a generic satellite in LEO subjected to a collision from different directions and with different velocities. This paper presents the results of agents trained with RL in training scenarios as well as in previously unknown situations using different methods such as DQN, REINFORCE, and PPO.

1 INTRODUCTION

This research aims to develop an algorithm for computing a collision avoidance manoeuvre followed by a return to the original orbit for a satellite with low-thrust propulsion. The algorithm takes the satellite's orbit, the space object's orbit, and the collision time as inputs while minimizing propellant usage. The problem was framed as a reinforcement learning (RL) task in a simulated environment.

The algorithms used for the task are the REINFORCE (Sutton and Barto, 2018) and PPO (Schulman et al., 2017) algorithms in the continuous action space setting, while the DQN (Mnih et al., 2013) algorithm is used in the discrete action space setting. The implementation of the environment[1], as well as the learning agent[2], along with experiments and results are made open source on *GitHub*.

To the authors' knowledge, this is the first work that attempts to optimize the collision avoidance manoeuvre and the return manoeuvre simultaneously using RL techniques. In principle, this allows the RL agents to compute overall more efficient solutions to the collision avoidance problem than if they were to analyze the two manoeuvre problems separately. As shown in the literature review, Section 2, previous research focused on solving just one aspect of the problem, making direct comparisons difficult. The environment developed in this paper is available for future research in academia and industry, providing a benchmark for further studies.

We summarize the most important contributions of this research as follows:

- An RL algorithm that is able to solve the CAM (Collision Avoidance Manoeuvre) optimization problem and the orbital change optimization problem (the return manoeuvre) simultaneously.
- An RL algorithm capable of solving the CAM optimization problem with both a discrete and a continuous action space;
- An *OpenAI gym* environment architecture and implementation suitable for simulations of satellite conjunctions, for future research in academia and industry.

The paper is structured as follows. The next section presents previous work on the topic under discussion. Section 3 describes the proposed new methods. The evaluation and discussion of the results of these approaches is presented in Section 4. In the last sec-

[1] https://github.com/AlexSolomon99/SatColAvoidEnv
[2] https://github.com/AlexSolomon99/SatColAvoidance

Solomon, A. and Paduraru, C.
Collision Avoidance and Return Manoeuvre Optimisation for Low-Thrust Satellites Using Reinforcement Learning.
DOI: 10.5220/0013249000003890

In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 3*, pages 1009-1016
ISBN: 978-989-758-737-5; ISSN: 2184-433X

tion, conclusions and possible perspectives for future work are presented.

2 RELATED WORK

One of the most common traditional approaches for trying to compute an optimal low-thrust manoeuvre is to divide the projected trajectory into arcs, followed by an individual optimization of each arc, as mentioned in (D. M. Novak, 2011) or (Whiffen, 2006). As detailed in (Tipaldi et al., 2022), there are many tasks related to satellite control that are addressed with RL approaches, such as Spacecraft systems touching down on extraterrestrial bodies (Gaudet et al., 2020; B. Gaudet, 2020), GTO/GEO transfers (Holt et al., 2021), interplanetary trajectory design (Zavoli and Federici, 2021), and even constellation orbital control (Yang et al., 2021).

2.1 RL Based CAM Optimization

In (Pinto et al., 2020), a Bayesian deep learning approach using RNNs predicts collision probability (PoC) from conjunction data messages (CDMs). A similar method using LSTMs and GRUs to predict event risk was explored in (Boscolo Fiore, 2021), but with limited success. In (N. Bourriez, 2023), a Deep Recurrent Q-Network (DRQN) was used to compute collision avoidance maneuvers (CAMs) in simulated conjunctions. The approach, involving partial observations and discrete thrust actions for 3D maneuvers, showed consistent improvement in the Huber loss across episodes, though cumulative rewards were not quantitatively measured.

2.2 RL-Based Optimization of the Orbital Transfer

In orbital transfer optimization, the agent is rewarded based on how close the current trajectory is to the target one. (LaFarge et al., 2021) explores closed-loop steering in dynamic multi-body environments, while (Casas et al., 2022) trains an agent to operate MEO satellite thrusters for pericenter lifting using the Advantage Actor Critic algorithm. (Kolosa, 2019) successfully addresses three orbital change tasks—general orbital change, semimajor axis change, and inclination change—using the Deep Deterministic Policy Gradient (DDPG) algorithm, with observations based on equinoctial orbital elements.

3 METHODS

3.1 Environment Description

This article develops an agent to control a satellite's thrusters for collision avoidance and return to the original orbit. The problem is framed as an RL task, with the agent learning through interactions in a simulated environment based on OpenAI's *gymnasium* API[3] and inspired by similar work[4] (Casas et al., 2022).

The environment simulates a low-Earth orbit region where a satellite with low-thrust propulsion faces a collision. General parameters are detailed in Table 1, and the agent-environment interaction is explained in the following subsections.

3.1.1 Episode Setup

The general idea of the setup is that the collision setting is created first. The position of the satellite is created at a specific location, then the location of the piece of debris is created in its vicinity, to ensure the collision, and then their states are propagated backwards. These 3 processes are described below.

Satellite Orbit Creation

The orbit of the satellite is created by defining the 6 Keplerian elements and the corresponding time epoch that uniquely describes the position of the satellite in time and space. The default values for the first 5 Keplerian elements (i.e. excluding the true anomaly υ) are by default set to the values specified in the Table 1. The true anomaly (υ) and the corresponding time epoch are randomly generated so that the agent can learn its task regardless of the collision position or time.

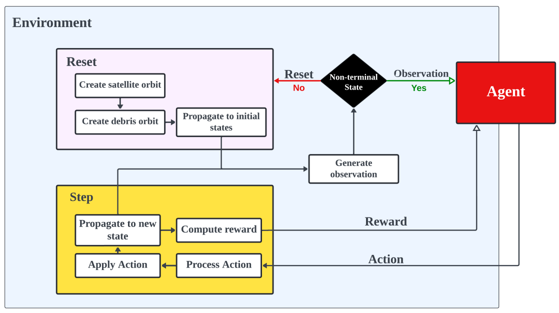

Figure 1: Schematic interaction between the Agent and the Environment.

[3] https://gymnasium.farama.org/index.html

[4] https://github.com/zampanteymedio/gym-satellite-trajectory

Table 1: Environment Parameters.

	Name	Value	Unit
Satellite Parameters	SMA (a)	6795	[km]
	e	0.18	-
	i	21.64	[°deg]
	ω	22.02	[°deg]
	Ω	60.0	[°deg]
	υ	0.0	[°deg]
	Mass	100.0	[kg]
	Surface area	1.0	[m^2]
	Reflection index	2.0	-
	Thruster force	0.1	[mN]
	Thruster I_{sp}	4000.0	[s]
Debris Parameters	Debris mass	10.0	[kg]
	Debris surface area	1.0	[m^2]
	Debris reflection index	2.0	-
Propagation Parameters	Method	Dormand Prince	
	Integrator min step	1.0	
	Integrator max step	200.0	
	Error threshold	1.0	[m]
	Perturbations	Not Applied	
Scenario Parameters	Time of collision	2023/06/16 - 00:00:00.0	[UTC]
	Time to avoid collision	4.0	[days]
	Collision Min Distance	2000.0	[m]
	Time to return to initial orbit	4.0	[days]
	Time step size	600.0	[s]

Piece of Debris Orbit Creation

The position vector $\vec{r}_d$ and the velocity vector $\vec{v}_d$ of the debris piece, components of the state vector, are generated using the state vector of the target satellite at the TCA:

$$\begin{aligned} \vec{r}_d &= \vec{r}_t + 10.0 \cdot \vec{r}_\Delta; & \vec{r}_\Delta \sim \text{Uniform}(0,1)^3 \\ \vec{v}_d &= -\vec{v}_t + 10.0 \cdot \vec{v}_\Delta; & \vec{v}_\Delta \sim \text{Uniform}(0,1)^3 \end{aligned} \tag{1}$$

where $\vec{r}_t$ and $\vec{v}_t$ are the position vector and the velocity vector of the target satellite at the TCA. The velocity of the debris is chosen to be opposite to the target satellite to avoid a long encounter collision.

Propagation to the Initial State

A numerical propagator is created for each of the 2 space objects, the target satellite and the piece of debris. The propagator is created using the parameters in the Table 1.

The initial state of the target satellite is set to four days before the TCA. Therefore, the propagator associated with the target satellite is used to propagate the state at the TCA of the target satellite 4 days into the past. This total duration is discretized in time by using the time step size mentioned in the Table 1.

Similarly, the initial state of the piece of debris is propagated backwards. The position of the piece of debris is only relevant in the vicinity of the collision state. Therefore, the state of the piece of debris is only propagated 40 minutes into the past, using the same time step size as in the case of the target satellite.

3.1.2 Observation Generation

The observation represents the information that the agent receives from the environment based on the current state. The observation is made up of four main components:

- State of the target satellite object;
- State of the piece of debris;
- Time until collision;
- Amount of fuel consumed;

The encoding of the information related to these four components is given to the agent using different approaches, with the differences between the two shown in the Table 5.

3.1.3 Actions

In each step, the agent has the option of controlling the thrusters of the target satellite, which are fired for the entire duration of the time step at the thrust level selected by the agent. The thrust vector is projected onto the three orthogonal axes that define the reference frame (either the GCRF or VNC, as given in the Table 5).

3.1.4 Reward System

The reward system is one of the most important things in a reinforcement learning setup, as it specifies the goal that the agent must achieve. The goals on which the rewards are based are the following:

1. **Collision Avoidance.** The minimum distance between the target satellite and the debris must be greater than a certain threshold value Δr_c.
2. **Return to Original Orbit.** The maximum difference between the final Keplerian elements and the initial elements must be smaller than a set of thresholds (Δr_a, Δr_e, Δr_i, Δr_ω, Δr_Ω).
3. **Fuel Consumption.** The fuel consumed by the satellite's engines must be minimized.
4. **Realism.** For the duration of the event, the satellite must follow orbits that are not too low (impact on Earth) and not too high (leaving LEO).

During the development phase, the values of the thresholds and the specific steps at which the rewards are returned have changed. However, the function that calculates the reward signal has the general form:

$$R = -(W_c \cdot r_c + R_{ret} + W_f \cdot r_f + R_{bound}) \tag{2}$$

The components of the reward signal are detailed below:

- W_c, r_c: The weight and reward contribution corresponding to (not) avoiding the collision:

$$\begin{aligned} r_c &= max(\Delta r_c - min(\|\vec{r}_{t,i} - \vec{r}_{d,i}\|), 0) \\ \Delta r_c &= 2000 \text{ [m]} \end{aligned} \tag{3}$$

  where "i" stands for all epochs in which the position vector of the debris is known. W_c varies depending on the approach, specific values are given in the Table 5.
- R_{ret}: The return corresponding to the (non-)return to the original orbit. 2 approaches have been defined:

  Approach-1:

$$R_{ret} = \sum_{k}^{(a,e,i,\omega,\Omega)} 0.2 \cdot C_k \tag{4}$$

$$C_k = \begin{cases} 0, & \text{if } \Delta_k < \Delta r_k \\ 1, & \text{Otherwise} \end{cases}$$

  Approach-2:

$$R_{ret} = \sum_{k}^{(a,e,i,\omega,\Omega)} W_k \cdot max(\Delta_k - \Delta r_k, 0) \tag{5}$$

$$W_k = \begin{cases} 0.002, & k = a \\ 0.01, & k = e \\ 0.01, & k = i \\ 0.1, & k = \omega \\ 0.001, & k = \Omega \end{cases}$$

  where Δ_k is the absolute difference between the current Keplerian element k and the original element. Δr_k is the corresponding threshold value for the k element. Specific values for Δr_k can be found in section 4, in the Table 5.
- W_f, r_f: The weight and premium contribution depending on fuel consumption:

$$\begin{aligned} r_f &= (M_i - M_c)/M_i \\ W_f &= 10000 \end{aligned} \tag{6}$$

  where M_i and M_c are the original and current mass of the satellite.
- R_{bound}: The return corresponding to leaving the orbital boundaries:

$$R_{bound} = \begin{cases} 0 & \text{, if } 6500 < \|\vec{r}_{target}\| < 8000 \text{ [km]} \\ 10 & \text{, otherwise} \end{cases} \tag{7}$$

3.2 Agent Implementation

The agent is an algorithm that is able to perform actions in certain states and improve the quality of these actions based on the rewards received from the environment. There are 3 algorithms that serve as agents in the environment described in the previous section: REINFORCE, DQN and PPO.

3.2.1 REINFORCE

REINFORCE is one of the simplest RL algorithms that can be implemented. It is an on-policy algorithm, i.e. the policy that is optimized is also the one used to select actions in training. In this scenario, the algorithm is used for the continuous action space, although it could have been used for the discrete action space.

The policy is a neural network which outputs the mean (μ) and standard deviation (std) of the distributions from which the action components are selected.

The discount factor γ is a measure of how much relevance current actions have for future rewards and the number of training steps represents the number of episodes used for training before an episode is used purely for evaluation. The hyperparameters used for the REINFORCE algorithm are specified in the Table 2:

Table 2: Hyperparameters used in the REINFORCE algorithm.

Hyperparameter Name	Value
Discount factor γ	0.99
Number of training steps	5
Hidden Layer 1 size	128
Hidden Layer 2 size	64
Optimiser	Adam
Learning rate	0.01

3.2.2 Deep Q-Network

The Deep Q-Network (DQN) algorithm is an off-policy algorithm, in contrast to the previously presented one. DQN uses a ε greedy approach for selecting actions during training. It is also an algorithm that can only be used in the context of discrete actions:

$$a = (a_1, a_2, a_3), \text{ with } a_i \in \{-1, 0, 1\} \quad (8)$$

In the DQN implementation, both the policy network (Q-net) and the target network are neural networks with identical architectures but different parameters. They output the expected reward for each action. A key feature in the current implementation is the addition of a normalization layer. In Approach-1, observations were normalized in the environment, while in Approach-2, normalization was handled by a normalisation layer added to the network.

The replay memory (R_M) is used, which stores experiences in a double-ended queue ("deque") and randomly selects them for model updates. To stabilize training, the gradients of the Q-net parameters are truncated to prevent excessive gradient norms. The hyperparameters used in the implementation of the DQN algorithm for the 2 different approaches are listed in the Table 3.

3.2.3 Proximal Policy Optimization

The PPO algorithm uses a combination of a policy gradient algorithm, called Actor, and a value-based algorithm, called Critic. The PPO algorithm implemented in this article is used in a continuous space and follows the implementation developed in (Schulman et al., 2017). The reason for choosing PPO is its stability in the training process, which results from the

Table 3: Hyperparameters used in the DQN algorithm.

Hyperparameter Name	Value Apr. 1	Value Apr. 2
Action space	[-1, 0, 1]	[-1, 0, 1]
Batch size	128	128
Replay Memory size	10000	10000
Discount factor γ	0.99	0.99
ε_{start}	0.9	0.9
ε_{end}	0.05	0.05
ε_{decay}	1000	100000
τ	0.005	0.005
Norm Layer	No	Yes
Hidden Layer 1 size	500	128
Hidden Layer 2 size	200	64
Optimiser	Adam	Adam
Learning rate	0.0001	0.0001
Clip Norm	5	5

update strategy used for the actor. The neural network used to represent the actor (the policy) is designed using the hyperparameters given in the Table 4.

Table 4: Hyperparameters used in the PPO algorithm.

Hyperparameter Name	Value
Steps per epoch	5 x 1151 (5 episodes)
Discount factor γ	0.99
GAE λ factor	0.97
Maximum KL	0.025
Maximum optimisation steps	80
Clip ratio ε	0.2
Learning rate actor	0.0003
Learning rate critic	0.001
Actor Hidden Layer 1 size	128
Actor Hidden Layer 2 size	64
Critic Hidden Layer 1 size	64
Optimiser	Adam

4 EVALUATION

This section presents the results obtained by training the three RL algorithms presented in the previous section with the two main approaches developed. Approach-1 represents the first set of parameters that showed relevant results, while Approach-2 represents the final set of parameters that showed the best results in the more restrictive setting. Approach-2 is an improvement over Approach-1. The main differences between the two approaches are described in the Table 5.

The models are compared both quantitatively and qualitatively. The evaluation of an agent includes:

- **Raw Reward Sum:** The sum of the rewards received for the duration of an episode. This value is averaged over 30 independent runs.

- **Collision Avoidance Status:** The collision must be avoided under all circumstances.
- **Return to Original Orbit:** The evolution of the first 5 Keplerian elements is analyzed quantitatively and qualitatively. The agent returns to the original orbit if all elements in the final state are within the specified threshold values.
- **Fuel Consumption:** If all previous conditions are met, a model with lower fuel consumption is preferred.
- **Action Choices:** The agent that performs the fewest actions and has a smoother action change is preferred.

4.1 Approach-1 Results

The algorithms REINFORCE, DQN and PPO have been trained and the test results are available in the Table 6. The columns "Collision Avoided" and "Returns to Orbit" show the percentage of episodes in which the agent reached the corresponding target, while the column "Fuel Used" shows the percentage of fuel used out of the total amount that could potentially have been used.

The table shows that the DQN algorithm performed best of the three algorithms. However, the goals that had to be achieved were not very restrictive for Approach-1 - see Table 5. The thresholds defining the initial orbit are rather loose, which is one of the reasons why the model seemed to perform so well. In fact, not all episodes return to the initial orbital elements - in some cases the elements deviate further from the initial elements. However, since the thresholds are not very restrictive, the final orbit was still within the appropriate range to be called equivalent to the initial orbit. Approach-2 addresses some of the issues observed with this first approach.

4.2 Approach-2 Results

In Approach-2, emphasis was placed on the task of returning to the initial orbit. The steps at which the return was rewarded (or penalized) were increased and the thresholds were significantly lowered (see Table 5) to penalize excessive deviations. In addition, the reward contribution of the return to the initial orbit component has been changed to be optimizable instead of a step function (Equation 5). This is all done to incentivize the agent to return to the initial orbit.

The observation structure was changed, as was the reference frame of the manoeuvre presentation, as described in the Table 5. The test results obtained with Approach-2 are shown in the Table 7. The results are not directly comparable with those of

Table 5: Implementation differences between approaches.

Element	Approach-1	Approach-2
Satellite state representation	Cartesian Coordinates - GCRF	Keplerian Elements
Piece of debris representation	Sequence of State-Vectors	Estimated min. distance (between satellite and debris)
Manouevre Thrust Components	GCRF ref. frame	VNC ref. frame
Coll. Avoid. reward weight (W_c)	0.01	0.0005
Return to Orbit Thresholds		
Semi-major axis (Δr_a)	500.0 [m]	10.0 [m]
Eccentricity (Δr_e)	0.001	$1e$-8
Inclination (Δr_i)	1.0 [rad]	$1e$-8 [rad]
Argument of Perigee (Δr_ω)	10.0 [rad]	10.0 [rad]
RAAN (Δr_Ω)	1.0 [rad]	$1e$-8 [rad]

Table 6: Agents comparison averaged over 30 episodes - Approach-1.

	Rewards Sum	Rewards Std	Coll. Avoided [%]	Orbit Return [%]	Fuel Used [%]
REINF.	-3.28	**0.10**	**100**	70	84.18
DQN	**-0.68**	0.19	**100**	**100**	**16.6**
PPO	-2.34	0.74	**100**	3	22.62

Table 7: Agents comparison averaged over 30 episodes - Approach-2.

	Rewards Sum	Rewards Std	Coll. Avoided [%]	Orbit Return [%]	Fuel Used [%]
REINF.	-1301.48	67.22	**100**	0	78.29
DQN	-258.51	74.44	**100**	0	52.83
PPO	**-175.23**	**11.59**	**100**	0	**38.49**

Approach-1, as the reward system and the observation structure differ significantly. The task of returning to the original orbit was also not fulfilled by all models in all episodes performed, which is expected given the extremely strict thresholds. Qualitatively, all agents trained according to Approach-2 performed better than the agents trained according to Approach-1.

Just looking at the numbers, it looks like the PPO model achieves the best overall results with Approach-2. It has the highest reward sum over the evaluation episodes, the smallest standard deviation and the lowest fuel consumption. However, when looking at the evolution of the Keplerian elements and the action profiles (Figures 2 - 5), the DQN model is the clear favorite. The development of the Keplerian elements follows the same pattern in almost all episodes as in Figure 2 for the DQN model: a clear, steady increase in the absolute magnitudes of the differences to the initial elements, followed by a steady decline, almost to the same starting point. Looking at the action profiles in Figure 3, there are also no rapid fluctuations or long periods of fluctuation. These rapid fluctuations are however present in the manoeuvres computed by the PPO model. This is expected, given the model operates in the continuous action space setting. Even though the PPO model obtains the best results quantitatively, the sequence of manoeuvres given by the model are unpractical.

5 CONCLUSIONS

This paper presents an algorithm for collision avoidance and return manoeuvres for a low-thrust satellite in Low-Earth orbit, using reinforcement learning (RL). A simulated environment allowed the agents to learn from different states and optimize thruster control to avoid collisions, return to the original orbit, and minimize fuel consumption.

Three algorithms were tested: REINFORCE, PPO (for the continuous action space), and DQN (for the discrete action space). Two approaches were explored. In Approach-1, satellite states and actions were given in the GCRF reference frame, and the agent received sparse rewards. Approach-2 used Keplerian elements, frequent rewards, and stricter goals. All algorithms avoided collisions, but returning to the initial orbit was challenging, especially in the continuous action space. The PPO algorithm favored fuel efficiency, while DQN succeeded in both tasks, though often at a higher fuel cost. The discrete action space formulation proved easier to implement and more effective for the task.

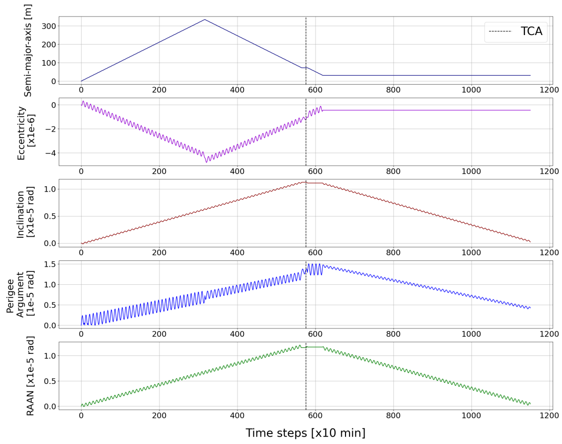

Figure 2: Keplerian Elements Variation - DQN Approach-2 (Best Model Qualitatively).

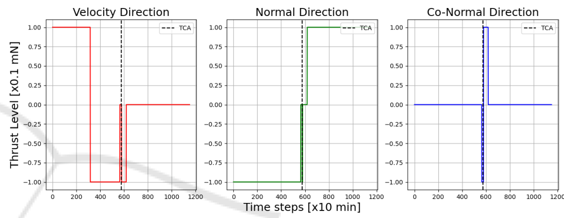

Figure 3: Agent Actions - DQN Approach-2.

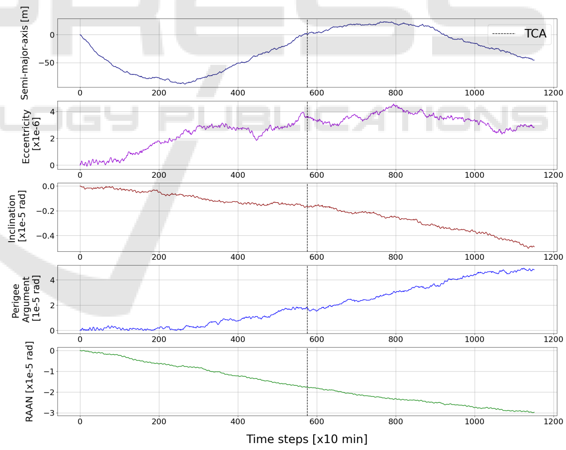

Figure 4: Keplerian Elements Variation - PPO Approach-2 (Best Model Quantitatively).

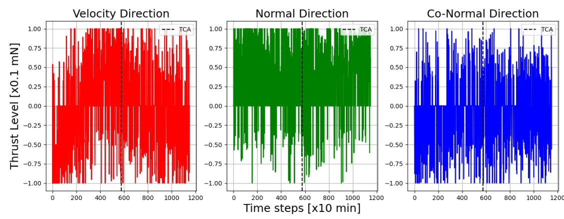

Figure 5: Agent Actions - PPO Approach-2.

Ultimately, the DQN algorithm in Approach-2 provided the best results, showing that RL can effectively optimize collision avoidance and return manoeuvres for low-thrust satellites.

ACKNOWLEDGMENT

We thank the International Astronautical Congress, IAC 2024, Milan, Italy, October 14-18, 2024, for feedback offered on a preliminary form of this work. This research is partially supported by the project “Romanian Hub for Artificial Intelligence - HRIA”, Smart Growth, Digitization and Financial Instruments Program, 2021-2027, MySMIS no. 334906 and a grant of the Ministry of Research, Innovation and Digitization, CNCS/CCCDI-UEFISCDI, project no. PN-IV-P8-8.1-PRE-HE-ORG-2023-0081, within PNCDI IV.

REFERENCES

B. Gaudet, R. Linares, R. F. (2020). Six degree-of-freedom body-fixed hovering over unmapped asteroids via lidar altimetry and reinforcement meta-learning. *Acta Astronautica*, 172:90–99.

Boscolo Fiore, N. (2021). *Machine Learning based Satellite Collision Avoidance strategy*. PhD thesis, Politecnico Milano.

Casas, C. M., Carro, B., and Sanchez-Esguevillas, A. (2022). Low-thrust orbital transfer using dynamics-agnostic reinforcement learning.

D. M. Novak, M. V. (2011). Improved shaping approach to the preliminary design of low-thrust trajectories. *Journal of Guidance, Control, and Dynamics*.

Gaudet, B., Linares, R., and Furfaro, R. (2020). Adaptive guidance and integrated navigation with reinforcement meta-learning. *Acta Astronautica*, 169:180–190.

Holt, H., Armellin, R., Baresi, N., Hashida, Y., Turconi, A., Scorsoglio, A., and Furfaro, R. (2021). Optimal q-laws via reinforcement learning with guaranteed stability. *Acta Astronautica*, 187:511–528.

Kolosa, D. S. (2019). *A Reinforcement Learning Approach to Spacecraft Trajectory Optimization*. PhD thesis, Western Michigan University.

LaFarge, N. B., Miller, D., Howell, K. C., and Linares, R. (2021). Autonomous closed-loop guidance using reinforcement learning in a low-thrust, multi-body dynamical environment. *Acta Astronautica*, 186:1–23.

Mnih, V. et al. (2013). Playing atari with deep reinforcement learning. *https://arxiv.org/abs/1312.5602*.

N. Bourriez, A. Loizeau, A. F. A. (2023). Spacecraft autonomous decision-planning for collision avoidance : a reinforcement learning approach. *74th INTERNATIONAL ASTRONAUTICAL CONGRESS (IAC)*.

Pinto, F. et al. (2020). Towards automated satellite conjunction management with bayesian deep learning. *Proceedings of NeurIPS 2020, AI for Earth Sciences Workshop*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *Arxiv.org/abs/1707.06347*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition.

Tipaldi, M., Iervolino, R., and Massenio, P. R. (2022). Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges. *Annual Reviews in Control*, 54:1–23.

Whiffen, G. (2006). Mystic: Implementation of the static dynamic optimal control algorithm for high-fidelity, low-thrust trajectory design. *Proceedings of AIAA/AAS Astrodynamics Specialist Conference and Exhibit*.

Yang, C., Zhang, H., and Gao, Y. (2021). Analysis of a neural-network-based adaptive controller for deep-space formation flying. *Advances in Space Research*, 68(1):54–70.

Zavoli, A. and Federici, L. (2021). Reinforcement learning for robust trajectory design of interplanetary missions. *Journal of Guidance, Control, and Dynamics*, 44(8):1440–1453.