

# LEO-GYM: A Reinforcement Learning Library for Satellite Control in LEO

Nektarios Aristeidis Tafanidis\* Avijit Banerjee\*  
George Nikolakopoulos\*

\* Robotics and AI group, Luleå University of Technology  
(email: {nektaf, aviban, geonik}@ltu.se).

## Abstract:

Motivated by recent advances in Reinforcement Learning (RL) and the lack of open-source tools for training and benchmarking satellite guidance and control, we introduce **LEO-GYM**: a lightweight Python library for formulating RL problems for satellites in Low Earth Orbit (LEO). The framework decomposes problems into three classes, the low-level dynamics, the training environment and a satellite object that bridges them. LEO-GYM enables the creation of custom scenarios without imposing rigid class hierarchies. We present the architecture, key components, and an illustrative orbit-correction task modeled as a semi-Markov decision process. LEO-GYM is released as open-source to support and foster reproducible research in autonomous space operations.

Copyright © 2025 The Authors. This is an open access article under the CC BY-NC-ND license (<https://creativecommons.org/licenses/by-nc-nd/4.0/>)

*Keywords:* Low Earth Orbit, Reinforcement Learning, Python Library

## 1. INTRODUCTION

In recent years, RL has gained a lot of popularity amongst researchers from multiple disciplines, ranging from research into algorithm development as well as to applied fields such as robotics, economics, cybersecurity, etc. Although RL is a powerful tool, setting up a RL problem is very time-consuming, not only from the formulation perspective but also implementation wise. For popular robotics platforms like Unmanned Aerial Vehicle (UAVs), manipulators or quadrupeds, there is a plethora of simulation software available online. A few notable examples include Isaac Gym and Isaac Lab Makoviychuk et al. (2021); Mittal et al. (2023), OmniDrones Xu et al. (2023) and PyBullet Coumans and Bai (2016–2019). Such libraries allow for researchers to easily set up experiments and test and develop their algorithms, without spending time in implementation and debugging which can be very time-consuming especially in RL, where pinpointing the exact issue in the environment-algorithm loop is not straightforward. Most importantly, they also provide a common benchmark for comparing and reproducing results.

However, for satellite systems, the resources are quite limited. Closest prior work is BSK-RL Stephenson and Schaub (2024), which uses the Basilisk high-fidelity spacecraft simulator to study discrete spacecraft tasking under realistic bus constraints and multi-satellite coordination. Agents interact through flight-software modes, enabling research on imaging/downlink/charging decisions with modeled subsystem failures and resource limits.

\* The LEO-GYM repository is publicly available at: [https://github.com/NekTaf/leo\\_gym](https://github.com/NekTaf/leo_gym)

\*\*This work has been partially funded by the European Space Agency (ESA) open Invitations to Tender (ITT) and innovation research grant in OPTACOM project, in collaboration with OHB Sweden under Grant Contract no: OPC-OSE-CC-0536

In this paper we present LEO-GYM, a Python-based library, which addresses a different layer of satellite autonomy, that of thruster-level orbital guidance and control for satellites in LEO. LEO-GYM can be used to train policies to conduct maneuvers such as for orbit correction and collision avoidance with external debris. Motivations behind this is both the nature of the available on-board hardware, where running in real-time computationally heavy optimization algorithms is oftentimes infeasible, as well as the difficulties associated with designing analytical solutions for complex mission scenarios.

The paper is organized as follows:

- Section 2: Overview of the library architecture.
- Section 3: Overview of the implemented dynamics.
- Section 4: Key concepts and terminology in RL.
- Section 5: Step-by-step tutorial on a satellite orbit-correction problem in LEO-GYM.
- Section 6: Future directions and features under development.

## 2. LIBRARY STRUCTURE

The core idea behind LEO-GYM is to provide a lightweight and flexible framework for training policies for satellite guidance and control. LEO-GYM currently implements orbital translational dynamics and has been demonstrated in orbital correction and collision avoidance with external debris scenarios Tafanidis et al. (2025c,b,a). The library can be divided into the following three major classes:

- **Dynamics**—At the core of LEO-GYM used for defining the satellite dynamics and propagating its states
- **Environment**—Inherits from FARAMA Gymnasium Towers et al. (2024), and defines the environment which will interact with the agent to train the pol-

icy, computing the reward, defining the observations, actions and termination/truncation conditions.

- **Satellite**—Acts as a bridge between the dynamics and the environment. Here conversions between different reference frames, interactions between different objects (ex. chief-deputy satellite or satellite-debris), maneuver types, logging and saving of satellite states, among other things, are defined.

A high level diagram of LEO-GYM's structure is presented in fig. 1. The library is designed in a way to give the user freedom to design custom scenarios without imposing a strict inheritance structure and hierarchy. The user should be able to easily introduce and implement new state transformations, create custom maneuvers with different constraints and introduce multiple space objects in the simulation.

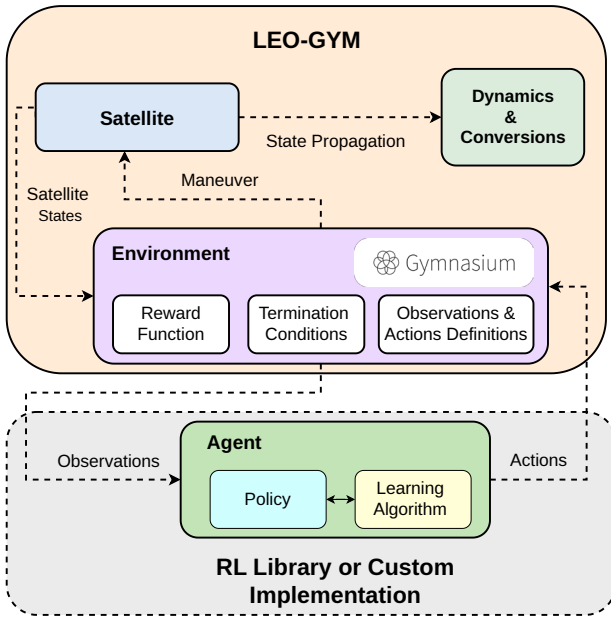


Fig. 1. Block diagram of LEO-GYM. The **Satellite** object acts as the communication bridge between the **Environment** and RL agent with the low-level dynamics. The **Satellite** object converts the maneuvers plans and calls the **Dynamics** to propagate the states, while the **Environment**, converts the satellite data into policy observations and training signals (ex. reward) to be used by the agent during training.

A basic **Satellite** class structure is presented below:

```
class Satellite():
    def __init__(self):
        """Initialize class"""
    def reset_sat_states(self)->None:
        """Initialize satellite states"""
    def apply_maneuver(self, manplan:np.ndarray)->None:
        """Access dynamics class and apply maneuver plan (manplan)"""
    def save_sat_states(self, path:str)->None:
        """Save data & configuration params to path"""
```

The key components of any **Satellite** class are a method for resetting the states, in order to reset an episode during

training, a method for applying the desired maneuver and propagating the states and a way to save states for logging purposes.

### 3. DYNAMICS

LEO-GYM implements a number of gravitational perturbations present in LEO. These include the irregular Earth gravity, third body gravitational perturbations from the moon and sun, Solar Radiation Pressure (SRP), and atmospheric air drag. A detailed overview of the perturbation dynamics is given in the subsections below. In the equations below  $\mathbf{r}, \mathbf{v}$  represents the satellite position and velocity vectors respectively.

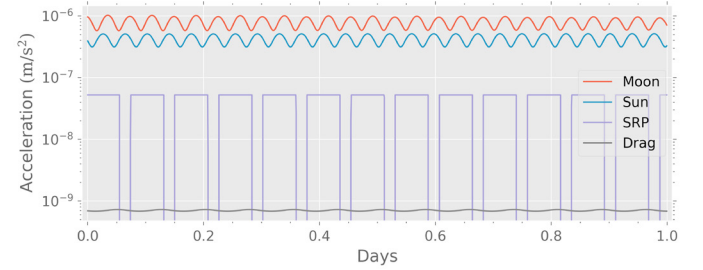


Fig. 2. Visualization of different perturbation induced accelerations on the satellite operating in LEO over one day's worth of simulation.

#### 3.1 Earth's Gravitational Perturbation:

The gravitational field can be described as a potential, whose gradient provides acceleration due to gravity. The higher degree spherical harmonics gravity perturbing accelerations are given as Vallado (2001)

$$\mathbf{w}_g = \nabla \left( \frac{\mu}{r} \sum_{n=2}^{\infty} \sum_{m=0}^n H_1 H_2 \right), \quad (1)$$

$$H_1 = \left( \frac{R_E}{\|\mathbf{r}\|} \right)^n, \quad (2)$$

$$H_2 = P_{nm}(\sin \varphi) (C_{nm} \cos m\lambda + S_{nm} \sin m\lambda), \quad (3)$$

where  $n$  and  $m$  are the degree and order of spherical harmonics,  $R_E$  is the Earth's radius,  $P_{nm}$  is the Legendre functions with argument  $\sin \varphi$ ,  $\varphi$  and  $\lambda$  are the respective geocentric latitude and longitude of the satellite and  $C_{nm}$  and  $S_{nm}$  are the harmonic coefficients of the potential. The Earth's gravity potential can be modeled with very high accuracy, using the higher degree and order Earth Gravitational Model (EGM)96. The Gravity Recovery and Climate Experiment (GRACE) model GGM03C provides the data for spherical harmonic coefficients complete to the degree and order 360.

#### 3.2 Atmospheric Drag:

The drag acceleration experienced in LEO is calculated according to

$$\mathbf{w}_{\text{drag}} = -\frac{1}{2} \rho \frac{C_d A_d}{m} \|\mathbf{v}_{\text{rel}}\|^2 \frac{\mathbf{v}_{\text{rel}}}{\|\mathbf{v}_{\text{rel}}\|}. \quad (4)$$

where  $\mathbf{v}_{\text{rel}} = \mathbf{v} - \boldsymbol{\omega}_{\oplus} \times \mathbf{r}$  is the spacecraft velocity relative to the atmosphere,  $m$  the mass of the satellite. It is assumed

that the atmosphere rotates at the same angular speed as the Earth ( $\omega_{\oplus}$ ). The atmospheric density at the altitude of the spacecraft is indicated by  $\rho$  and is modeled using a piecewise exponential density model,  $C_d$  is the drag coefficient, and  $A_d$  the flat plate area in nominal attitude.

### 3.3 Third Body Perturbation:

The third body gravity perturbations are exerted by the Sun and Moon. Thus, the gravity acceleration due to the third body effects can be expressed as Vallado (2001)

$$\mathbf{w}_{3b} = \mu_{\odot} \left( \frac{\mathbf{r}_{\odot s}}{\|\mathbf{r}_{\odot s}\|^3} - \frac{\mathbf{r}_{\odot \oplus}}{\|\mathbf{r}_{\odot \oplus}\|^3} \right) + \mu_{\ominus} \left( \frac{\mathbf{r}_{\ominus s}}{\|\mathbf{r}_{\ominus s}\|^3} - \frac{\mathbf{r}_{\ominus \oplus}}{\|\mathbf{r}_{\ominus \oplus}\|^3} \right), \quad (5)$$

where  $\mu_{\odot}$  and  $\mu_{\ominus}$  corresponds to the gravitational parameter of Sun and Moon respectively,  $\mathbf{r}_{\odot s} = \mathbf{r}_{\odot} - \mathbf{r}_s$  and  $\mathbf{r}_{\ominus s} = \mathbf{r}_{\ominus} - \mathbf{r}_s$  are the respective satellite to Sun and Moon position vectors, and  $\mathbf{r}_{\odot \oplus} = \mathbf{r}_{\odot} - \mathbf{r}_{\oplus}$  and  $\mathbf{r}_{\ominus \oplus} = \mathbf{r}_{\ominus} - \mathbf{r}_{\oplus}$  are the respective Earth to Sun and Moon position vectors. The positions of the Moon and Sun are calculated using the NASA SPICE toolkit Annex et al. (2020).

### 3.4 Solar Radiation Pressure (SRP):

The cannonball disturbance model and dual conical shadow model Montenbruck and Gill (2000) are considered for the SRP

$$\mathbf{w}_{\text{srp}} = -\nu P_{\text{srp}} C_r \frac{A_s}{m} \left( \frac{\text{AU}}{\|\mathbf{r}_{\odot s}\|} \right)^2 \frac{\mathbf{r}_{\odot s}}{\|\mathbf{r}_{\odot s}\|}, \quad (6)$$

where  $\mathbf{r}_{\odot s} = \mathbf{r}_{\odot} - \mathbf{r}_s$  the satellite to Sun vector,  $P_{\text{srp}}$  is the solar radiation pressure at 1 AU,  $C_r$  is the solar reflection coefficient,  $A_s$  is the exposed solar facing surface area of the satellite,  $\nu$  is the shadow function and is 0 if the satellite is in umbra, 1 if the satellite is in sunlight and between 0 and 1 if in penumbra.

## 4. REINFORCEMENT LEARNING

Before demonstrating the library, the formulation of a general RL problem will be presented. In RL, an **agent** interacts with an **environment** over discrete decision steps. At step  $t$ , given **state**  $\mathbf{s}_t$  the agent selects an **action**  $\mathbf{a}_t$  according to a stochastic policy  $\pi(\cdot; \theta)$  and receives from the environment a scalar **reward**  $r_t$  that evaluates the quality of said action with respect to an objective (e.g., how closely a satellite follows a desired trajectory).

During an RL episode, **termination** occurs when a terminal condition is met (e.g. successfully reached a goal or committed an undesired state violation), while **truncation** occurs when the agent does not solve the episode within a set amount of time.

Formally, the environment is modeled as a Markov Decision Process (MDP)  $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$  with state  $\mathbf{s}_t \in \mathcal{S}$ , action  $\mathbf{a}_t \in \mathcal{A}$ , transition kernel  $P(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$ , and reward function  $R(\mathbf{s}_t, \mathbf{a}_t)$  so that  $r_t = R(\mathbf{s}_t, \mathbf{a}_t)$  and trajectories are induced jointly by  $P$  and  $\pi(\cdot; \theta)$ .

### 4.1 Partial State Observability

Importantly, the input observation of the environment,  $\mathbf{o}_t$ , available to the policy need not coincide with the full underlying state  $\mathbf{s}_t$  of the system. This partial observability is common in practice, as for example, in an orbital correction scenario where the agent may only access relative states, while many other factors influencing its motion (environmental perturbations, absolute position, ephemeris time) remain unobserved. This corresponds to a Partially Observable Markov Decision Process (POMDP). For clarity, the observation notation  $\mathbf{o}_t$  as the policy input, will be used from now on.

### 4.2 Learning Objective

The learning objective is to find parameters  $\theta$  (i.e., the Neural Network (NN) weights) that maximize the expected discounted return over a finite horizon  $T$  and is defined as

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k, \quad (7)$$

where  $\gamma \in [0, 1)$  is a discount factor. A small  $\gamma$  makes the agent myopic (favoring immediate reward), while  $\gamma \rightarrow 1$  values distant rewards, extending the effective horizon but at the same time increasing the return variance and sensitivity to long-term estimation errors.

### 4.3 Variable Duration Actions

An interesting subfamily of decision models for RL is the class of Semi-Markov Decision Processes (SMDPs), where decisions have variable physical duration  $\tau_t > 0$  (e.g., the firing duration of the satellite thrusters can vary between environment transitions). This variable time  $\tau_t$  is often referred to as the sojourn time. As a result environment step-based discounting  $\gamma^k$  can bias the agent toward either short steps or long steps depending on the reward scale. Instead, the return should be discounted with regards to the real time

$$G_t = \sum_{k=t}^{T-1} \gamma^{u_k} r_k, \quad (8)$$

$$u_k = \sum_{i=t}^{k-1} \tau_i. \quad (9)$$

### 4.4 Proximal Policy Optimization

In order for the agent to learn the best policy, a learning algorithm will need to be used. One of the most popular ones is Proximal Policy Optimization (PPO). The policy is updated according to

$$L_{\pi}(\theta) = \mathbb{E}_t \left[ \min \left( \rho_t \hat{A}_t, \text{clip}(\rho_t, 1 \pm \varepsilon) \hat{A}_t \right) \right], \quad (10)$$

where

$$\rho_t = \frac{\pi(\mathbf{a}_t | \mathbf{o}_t; \theta)}{\pi(\mathbf{a}_t | \mathbf{o}_t; \theta_{\text{old}})}. \quad (11)$$

PPO is an Actor-Critic type algorithm and thus uses a value baseline  $V(\mathbf{o}; \phi)$  reduces variance. With variable durations, the TD error and Generalized Advantage Function (GAE) are defined as

$$\delta_t = r_t + \gamma^{T-t} V(\mathbf{o}_{t+1}; \phi) - V(\mathbf{o}_t; \phi), \quad (12)$$

$$\hat{A}_t = \sum_{k=t}^{T-1} (\gamma\lambda)^{u_k} \delta_k. \quad (13)$$

For brevity terminal indicators are omitted in the equations. Bootstrapping is masked across episode ends by setting  $V(\mathbf{o}_{t+1}; \phi) = 0$  at terminals. GAE uses time-aware decay with  $\lambda_t = \lambda^{T-t}$  so the smoothing horizon is also measured in real time.

## 5. DEMONSTRATION

In order to get a better understanding of LEO-GYM, an example trajectory correction scenario is formulated. Periodic trajectory corrections are required to be undertaken by satellites in a constellation to keep them within some allowed boundaries around their ideal orbits. The deviations experienced by the satellite are a result of the various perturbations present in LEO. This operation is often referred to as Station Keeping (SK).

In this example, a policy will be trained to make Across Track (ACT) corrections. The relative motion between the ideal and real satellite trajectory is modeled using the Relative Orbital Elements (ROE) introduced by D’Amico in D’Amico (2010). ACT corrections involve controlling the relative inclination vector  $a\delta\mathbf{i} = [a\delta i_x, a\delta i_y]$  (m) by applying thrust in the Normal direction in the RTN frame. The firing location on orbit denoted by the Argument of Mean Latitude (AOML)  $u$  (rad), is another important variable that will determine how the relative inclination vector changes.

The correction maneuver is modeled as a finite burn. The policy will output a continuous variable which will be discretized to select the firing direction  $\{-f_n, +f_n\}$  (mN) and the maneuver planning times which include the delay till the maneuver start time  $\Delta t_{\text{del}}$  (min) and the duration of the maneuver  $\Delta t_{\text{dur}}$  (min).

### 5.1 Observations, Actions and Rewards

The observation and action spaces can thus be defined as

$$\mathbf{o} = [u, a\delta i_x \times 10^{-3}, a\delta i_y \times 10^{-3}] \in \mathbb{R}^3, \quad (14)$$

$$\mathbf{a} = \{[-f_n, +f_n], \Delta t_{\text{del}}, \Delta t_{\text{dur}}\} \in \mathbb{R}^3. \quad (15)$$

Note that the policy inputs are rescaled to improve the numerical stability of the learning. The reward function provides the agent with a small negative reward, which decreases the closer it gets to the required relative inclination magnitude  $\|a\delta\mathbf{i}\|_{\text{req}}$ . If the satellite violates an upper relative inclination magnitude  $\|a\delta\mathbf{i}\|_{\text{cut-off}}$  the episode

terminates, and a large negative reward is provided. This helps speed up training by discouraging the agent from exploring states from which it may be difficult to recover its position. Finally a large positive reward is provided and the episode is terminated, once the agent successfully reduces its relative inclination to the required lower value  $\|a\delta\mathbf{i}\|_{\text{req}}$ . The complete reward function is defined as

$$R(\mathbf{o}, \mathbf{a}) = \begin{cases} -100 \text{ and Terminate,} & \text{if } \|a\delta\mathbf{i}\| \geq \|a\delta\mathbf{i}\|_{\text{cut-off}}, \\ -\log(\|a\delta\mathbf{i}\| + \epsilon), & \text{if } \|a\delta\mathbf{i}\| > \|a\delta\mathbf{i}\|_{\text{req}}, \\ +100 \text{ and Terminate,} & \text{otherwise.} \end{cases} \quad (16)$$

Since the actions have variable duration and reward discounting is done according to the real elapsed time, rather than discrete episode timesteps. The truncation condition is applied when the episode exceeds a predefined time duration (e.g. 3 days worth of orbit). Truncation is also a key factor in speeding up training, as the agent is allowed only a certain amount of time to solve an episode, thus avoiding extremely long or uninformative rollouts.

### 5.2 Dynamics and Satellite Object

To start off, the `Satellite` and `Dynamics` objects will need to be defined. A sample `Dynamics` configuration is defined below:

```
ideal_traj_params = DynamicsConfig(
    # Thrust applied in RTN or ECI frame
    flag_rtn_thrust=True,
    # Perturbation flags
    flag_mass_loss=True,
    flag_pert_moon=False,
    flag_pert_sun=False,
    flag_pert_srp=False,
    flag_pert_drag=False,
    flag_pert_irr_grav=True,
    eph_time_0=..., # Ephemeris time (s)
    m=150, # Initial mass (kg)
    f_max=18e-3, # Max thrust of on-board thruster(N)
    Isp=860, # Specific impulse (s)
    Ad=1.3, # Drag reference area (m²)
    Cd=2.2, # Drag coef
    Cr=1.3, # SRP reflectivity coef
    As=1.3, # SRP reference area (m²)
    mf=136) # Dry mass (kg)
```

In total two trajectories will need to be generated, one for the ideal unperturbed satellite and one for the real satellite that will be controlled by the policy. In the example above the ideal trajectory parameters are defined as `ideal_traj_params`.

An almost identical copy of `ideal_traj_params` can be made for the real or perturbed trajectory parameters, by setting the perturbation flags `flag_pert_...` to `True`. Exception is the `flag_pert_irr_grav` flag, which is present in both cases, as in this example effects from Earth’s irregular gravity are not considered as perturbations, due to the low available on-board thrust of the satellite. Both of these trajectories will be generated through the satellite class.

A `Satellite` configuration for the current example can be defined as seen below:

```

satellite_params = SatelliteConfig(
# Simulation duration and sampling time
dt = 60,
days = 3,
# Configuration classes for the dynamics
ideal_traj_params = ideal_traj_params,
pert_traj_params = pert_traj_params,
# Initial position-velocity state for ideal
trajectory
rv0 = np.array(...))

```

The `Satellite` will start by generating the ideal trajectory for 3 days worth of orbit with a starting position-velocity vector `rv0` at the start of an RL episode. It will then provide the API for propagating the satellite each step and extracting the current states for defining the observation of the policy, as well as plotting and saving data.

### 5.3 Environment

In the `Environment` configuration, the allowable action ranges are defined as (`high_action`, `low_action`). Note that the action boundaries for eq. (14) encode real onboard thruster constraints, such as the minimum value for  $\Delta t_{\text{del}}$  representing the slew rate for rotating the onboard thruster and values for  $\Delta t_{\text{dur}}$  representing the minimum and maximum physical thruster firing durations. A sample `Environment` configuration is presented below:

```

env_cfg = GymConfig(
# Action ranges
high_action = [15, 110, 41], # f_n, Delta t_del,
Delta t_dur
low_action = [-15, 10, 1],
# Starting tracking error in classical Orbital
Elements
Delta_oe_ranges = ..., # (6,2) array
# Satellite Initialization
satellite_params = satellite_params,
# Termination conditions
adi_norm_cutoff = 2000, # Relative inclination
magnitude (m)
adi_norm_target = 50, # Relative inclination
magnitude (m)
seed = ...) # Environment seed

```

The variable `Delta_oe_ranges` defines the min-max ranges, from which the initial trajectory deviations will be sampled from, in terms of the classical orbital elements

$$\Delta a, \Delta e, \Delta i, \Delta \Omega, \Delta \omega, \Delta \nu,$$

which correspond in order to the semimajor axis, eccentricity, inclination, Right Ascension of the Ascending Node (RAAN), Argument of Periapsis (AOP), and true anomaly. For ACT correction, only the inclination and RAAN will affect the relative inclination. Finally, the values for the termination condition variables are also defined (the upper cut-off  $\|a\delta i\|_{\text{cut-off}}$  and lower target  $\|a\delta i\|_{\text{targ}}$  relative inclination magnitudes). Once all the required configuration parameters are set, the environment can be defined as seen below:

```
env = StationKeepingGym(cfg=env_cfg)
```

In this example, policy is trained using a modified version of the PPO algorithm, from the StableBaselines3 library Raffin et al. (2021). An example of a basic environment loop is presented below:

```

agent = PPO.load("policy.zip")
obs, _ = env.reset()
while True:
action, _ = agent.predict(obs)
obs, rew, term, trunc, info = env.step(action)
if term or trunc:
break

```

### 5.4 Simulation Results

To deploy the policy the following controller activation logic Tapanidis et al. (2025b) is implemented:

#### Algorithm 1 Across Track Correction Strategy

```

if  $\|a\delta i\| \geq \|a\delta i\|_{\text{trig}}$  then
  Activate policy to set  $\|a\delta i\|$  to  $\|a\delta i\|_{\text{req}}$ 
else
  Satellite coasting until next  $\|a\delta i\|_{\text{trig}}$  violation
end if

```

This activates the policy only when  $\|a\delta i\|$  violates the upper boundary  $\|a\delta i\|_{\text{trig}} = 750$  m, allowing otherwise the satellite to coast unactuated. Once activated the policy pushes the satellite to the lower  $\|a\delta i\|_{\text{req}} = 50$  m boundary. The results for a one year simulation of ACT correction maneuvers are presented in fig. 3. A shorter maneuver sequence for more clarity is presented in fig. 4.

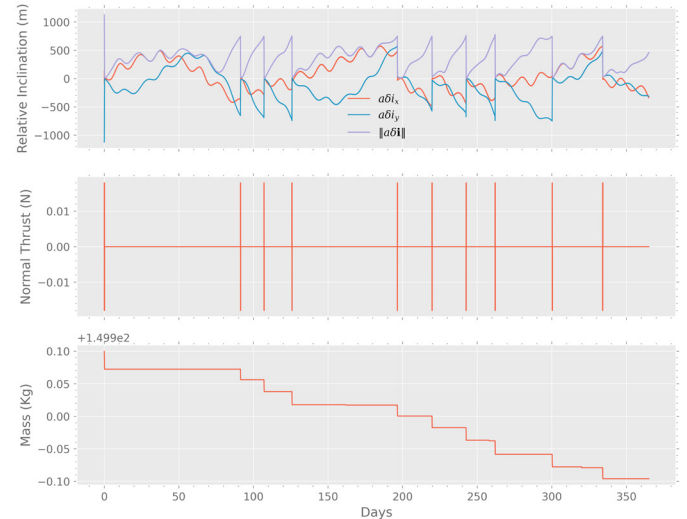


Fig. 3. Visualization of the relative inclination corrections for a one year simulation conducted in LEO-GYM. The trained policy is activated according to the logic defined in algorithm 1, to correct the relative inclination magnitude and reduce it to a lower “target” boundary, every time it violates an upper “trigger” boundary.

## 6. WORK IN PROGRESS

Current work on the library is mostly aimed in optimizing the execution speed. Currently LEO-GYM is implemented

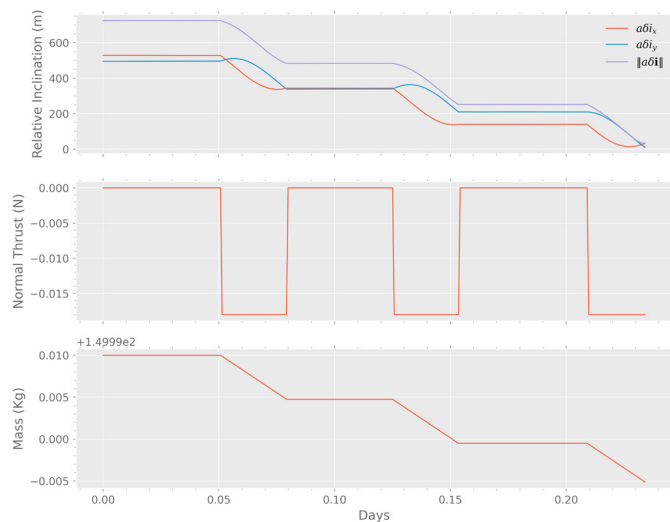


Fig. 4. Visualization of shorter sequence of maneuvers, equivalent to one successful RL episode.

in NumPy Harris et al. (2020). However, frameworks such as JAX Bradbury et al. (2018) can significantly speed up execution using Just In Time (JIT) compilation. Additionally, the library has focused so far on single agent examples. Future work will focus on creating tutorials and examples on multi-agent RL (such as coordinated formation-flying and joint collision-avoidance).

## 7. CONCLUSION

In this paper we presented LEO-GYM, a lightweight and flexible open-source Python library. Motivated by the lack of tools for modelling and solving satellite guidance and control problems with RL, the library enables researchers to set up scenarios, benchmark, and compare solutions. Its architecture is non-restrictive and allows the users interact directly with the core orbital-dynamics class by defining a custom satellite object that bridges the training environment/algorithm to the low-level dynamics (e.g., to define custom maneuvers and behaviors). As an illustrative example, we formulate and solve an orbit-correction task that trains a policy to compute finite-burn maneuvers to regulate relative orbital elements.

## REFERENCES

- Annex, A. et al. (2020). SpiceyPy: a pythonic wrapper for the SPICE toolkit. *Journal of Open Source Software*, 5(46), 2050. doi:10.21105/joss.02050. URL <https://doi.org/10.21105/joss.02050>.
- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs. URL <http://github.com/google/jax>.
- Coumans, E. and Bai, Y. (2016–2019). Pybullet, a python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>.
- D’Amico, S. (2010). *Autonomous formation flying in low earth orbit*. Ph.D. thesis, TU Delft.
- Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T.E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. doi:10.1038/s41586-020-2649-2. URL <https://doi.org/10.1038/s41586-020-2649-2>.
- Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., and State, G. (2021). Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470. URL <https://arxiv.org/abs/2108.10470>.
- Mittal, M., Yu, C., Yu, Q., Liu, J., Rudin, N., Hoeller, D., Yuan, J.L., Singh, R., Guo, Y., Mazhar, H., Mandekar, A., Babich, B., State, G., Hutter, M., and Garg, A. (2023). Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6), 3740–3747. doi:10.1109/LRA.2023.3270034.
- Montenbruck, O. and Gill, E. (2000). *Satellite Orbits*, volume 1. doi:10.1007/978-3-642-58351-3.
- Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268), 1–8. URL <http://jmlr.org/papers/v22/20-1364.html>.
- Stephenson, M.A. and Schaub, H. (2024). Bsk-rl: Modular, high-fidelity reinforcement learning environments for spacecraft tasking. In *75th International Astronautical Congress, Milan, Italy, IAF*.
- Tafanidis, N.A., Banerjee, A., and Nikolakopoulos, G. (2025a). Hybrid-action semi-markov reinforcement learning with shap-based explainability for leo collision avoidance and recovery. *Available at SSRN 5357485*.
- Tafanidis, N.A., Banerjee, A., Satpute, S., and Nikolakopoulos, G. (2025b). Reinforcement learning-based station keeping using relative orbital elements. *Advances in Space Research*.
- Tafanidis, N.A., Banerjee, A., Satpute, S.G., and Nikolakopoulos, G. (2025c). Reinforcement learning based intelligent control for low-thrust tight station keeping in low earth orbit. *International Journal of Control, Automation, and Systems*, 23(7), 2105–2116.
- Towers, M., Kwiatkowski, A., Terry, J., Balis, J.U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. (2024). Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*.
- Vallado, D.A. (2001). *Fundamentals of astrodynamics and applications*, volume 12. Springer Science & Business Media.
- Xu, B., Gao, F., Yu, C., Zhang, R., Wu, Y., and Wang, Y. (2023). Omnidrones: An efficient and flexible platform for reinforcement learning in drone control.