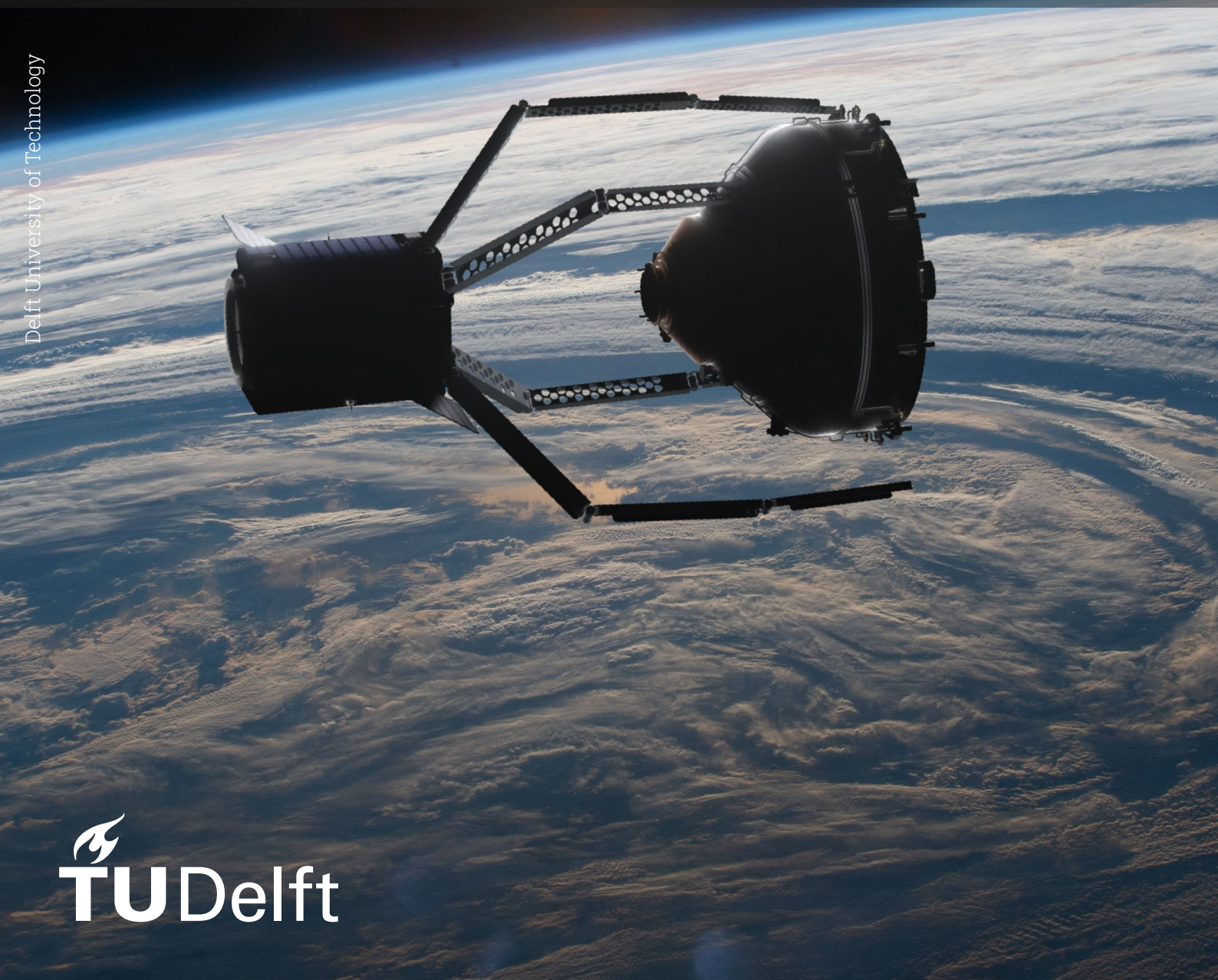
Robust Attitude Control in Active Debris Removal Missions using Reinforcement Learning

Master Thesis

M. R. Meijkamp



Delft University of Technology

TU Delft

Robust Attitude Control in Active Debris Removal Missions using Reinforcement Learning

Master Thesis

by

M. R. Meijkamp

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday August 30, 2024 at 9:30.

Student number:	4676718	
Thesis committee:	Dr. ir. E. Mooij,	TU Delft, chair
	Dr. J. Guo,	TU Delft, supervisor
	Dr. ir. E. van Kampen,	TU Delft, supervisor
	J. Liu,	TU Delft, supervisor
	Dr. S. Speretta,	TU Delft, independent assessor

Cover:	ClearSpace-1 captures Vespa - Courtesy of ESA/ClearSpace SA.
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `https://repository.tudelft.nl/`.

TU Delft

Preface

That's it, an end to a 7 year long journey of learning. Looking back on my time at TU Delft, I can honestly say that I could not have made a better decision in coming here to study. I have been challenged, but also inspired. I've learned it's okay to think big, to be excited about something, while also recognizing your own opportunities for improvement and the strength of working together.

During these years I've met many wonderful people, many who I can now call friends. I want to thank you for all the laughs, the motivation during the difficult moments, and especially the support during the thesis research. Whether it was a pep talk on the phone or the "we're all in this together"-mindset of our master workspace in 2.56, you made me determined to finish and made me a better person.

Thanks also to my parents, who gave me the freedom to choose any path I wanted, despite the crazy looks I sometimes got for trying to combine one thing too many. Thanks for keeping me sane and supporting me all the way.

Finally, thank you to my supervisors: Jian Guo, Erik-Jan van Kampen, and Jingyi Liu. Initially, I did not really know what I wanted. You gave me the opportunity to learn a completely new field, while still working on a space topic, with relevance to sustainability. I know I could sometimes be a bit overenthusiastic or a bit too optimistic, but you provided the right levels of criticism combined with confidence for me to take a step back, iterate, refine, and learn the most from the entire process. Thanks for steering me into the right directions and for asking the right questions.

To the reader, I learned so much while writing this thesis, and I hope you will too. It was fun to make, so I'm confident you will find the same excitement about this topic that I have found.

M. R. Meijkamp
Delft, July 2024

List of Figures

List of Tables

Nomenclature

α	Probability smoothing parameter
δ	Rigid-flexible coupling matrix
η	Modal coordinate
γ	Discount factor
$\mathcal{A}$	Action space
$\mathcal{S}$	State space
ω	Attitude rate
Φ	Shape function
ϕ	Error angle around the euler axis
π	Policy
σ	Standard deviation
τ	Polyak update coefficient
ε	Gyroscope noise
a	Action
b	Gyroscope bias
C	Damping matrix
c_a	Reward function action coefficient
c_p	Reward function non-improvement penalty coefficient
c_r	Reward function requirement coefficient
f_η	Natural frequency of the modal coordinates
J	Inertia tensor
K	Stiffness matrix
K_i	PID gain matrix i
M	External torques
o	Observation vector
p	Probability distribution of the next state in an MDP
Q	Action-value function
q	Attitude quaternion
r	Reward function
s	State
t	Time
u	Control torque
V	Value function
W	Wiener process
ADR	Active Debris Removal
DDPG	Deep Deterministic Policy Gradient
INDI	Incremental non-linear dynamic inversion
JIT	Just-In-Time
LEO	Low Earth Orbit
MDP	Markov Decision Process
PID	Proportional-integral-derivative
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RL	Reinforcement learning
SAC	Soft actor-critic
SQ	Sub-question
TD3	Twin-delayed DDPG
TD7	TD3 with four improvements
TRPO	Trust region policy optimization

Contents

1

Introduction, background, and Research Plan

1.1. Space debris and debris characterization

In-situ space activities require man-made objects in space. These activities have created a large number of space debris objects, defined as artificial non-functional space objects by Bernhard et al [1]. When the amount of objects becomes large enough and the objects are in similar enough orbits, the risk of collision between these objects becomes substantial. In 1978, Kessler first predicted that a chain reaction of collision events was possible and could result in a debris belt around the Earth, restricting access to space [2].

To understand and manage this threat, further knowledge is required of the characteristics and distribution of the debris environment. The assessment of this is an active field of research. Major contributions have been made by the ESA Space Debris Office [3] and NASA's Johnson Orbital Debris Program Office [4]. Schaub et al summarized three key points from those contributions [5]. Firstly, while the number of debris is increasing overall, the most rapid growth is concentrated in certain altitudes. Secondly, orbits with high inclination, particularly between 600-800 km and 1000-1500 km are populated most densely. Thirdly, the largest hazard to space missions is posed by smaller objects. This is because there are many of these smaller objects, they cannot be tracked, and are still very energetic. However, larger objects drive the growth of the number of debris objects.

A critical distinction in the characterization of space debris lies in the categorization between cooperative and non-cooperative debris. In this research, the definition by Shan et al [6] is used, where cooperative debris is classified using two distinct characteristics. Firstly, a debris object can be non-cooperative due to a lack of knowledge of the object in terms of physical parameters such as inertia, or orbital and attitude parameters. Secondly, the object can be non-cooperative due to a lack of capturability, which refers to a lack of docking locations or other physical complications in the capture process, such as a high (known) tumble rate. Consequently, non-cooperative debris objects present a more difficult challenge to understand and deal with appropriately.

1.2. Active Debris Removal

Historically, collision and break-up events are luckily still relatively rare. The collisions that did occur have been categorized and documented by NASA [7]. The number of debris objects in space is growing rapidly, as shown by figure 1.1. Of course, collisions are only part of the growth driver. However, Rossi and Valsecchi predicted in 2006 that in case no action is undertaken to mitigate the risk of collision, about 60 catastrophic collisions would happen over then next 100 years. This would roughly double the number of objects larger than 1 cm and represent a beyond-linear growth [8].

For this reason, various mitigation measures for space missions are being implemented, like those described by the IADC guidelines [10]. The guidelines focus on the "limitation of debris released dur-

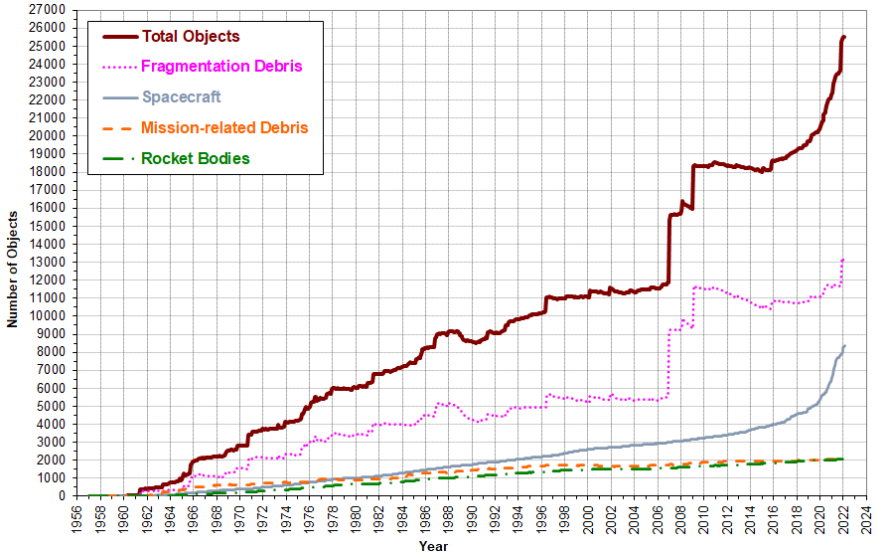


Figure 1.1: Monthly number of debris objects >10 cm in low Earth orbit by object type [9].

ing normal operations, minimisation of the potential for on-orbit break-ups, post-mission disposal, and prevention of on-orbit collisions". However, the guidelines only focus on new missions, which leaves the significant risk of the current debris environment unmitigated. For instance, Liou and Johnson have shown that the number of debris in low Earth orbit (LEO) will likely continue to grow even if no new missions are launched [11], which was verified by researchers at ESA [12]. In 2010, Liou wrote that active debris removal might be only option to reduce the number of debris to acceptable levels in the future [13]. Even though Liou's analysis only applies to the LEO environment, the paper establishes a clear need for further research into active debris removal missions. The research discussed in this report is intended to be a contribution to this body of research.

A wide variety of active debris removal methods have been conceived. A rough overview is outlined in figure 1.2. Other contactless methods have also been investigated, including laser systems [14] and ion beam shepherd systems [15]. Different studies and conceptual mission design processes have been executed in order to investigate the merits of different capturing methods.

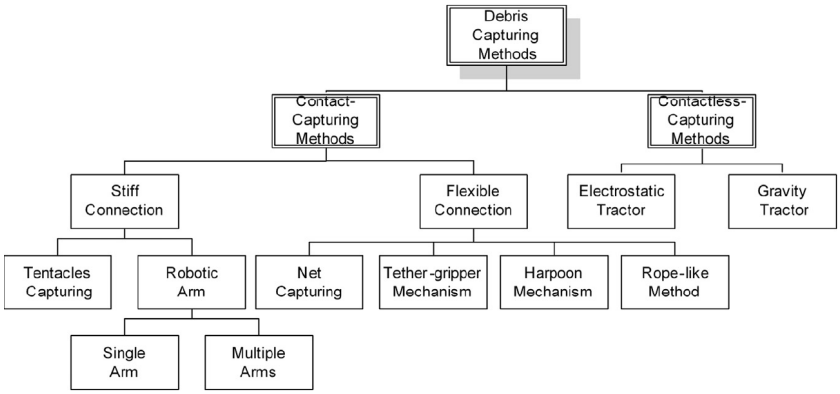


Figure 1.2: Concept diagram of capturing methods, from Shan et al [16].

One of the most influential of these studies was a pre-assessment study called e.Deorbit which was started in 2012 by ESA to lay out a technology roadmap and conduct preliminary systems design for an active debris removal mission [17]. In the context of this study, all industry proposals selected the robotic arm with a clamping mechanism as a baseline. The missions would rely based on expected heritage from DLR's proposed DEOS mission [18] [19]. However, the DEOS mission got cancelled in 2018. Nonetheless, the concept of robotic space manipulators is a space-proven technology [20]. Alterna-

tively, the RemoveDEBRIS mission led by Surrey Space Centre, was the first mission to successfully demonstrate net-capture and harpoon capture in-orbit [21]. While the demonstration of the capture technologies was successful, the planned demonstration for deorbiting the mission failed, and they did not attempt to control or actively de-orbit the net-captured target post-capture [22]. Other in-orbit demonstrator missions include Astroscale's ELSA-d mission, demonstrating capture, diagnosis, and client search technologies based on a magnetic docking mechanism [23], or their conceptual missions ADRAS-J (approach and inspect a large debris target) and COSMIC (remove two UK registered space debris targets) [24]. Recently, after finalization of the e.Deorbit study, ESA entered into a collaboration with ClearSpace for the ClearSpace-1 mission. This mission is aiming to use a servicer equipped with four tentacle robotic arms to capture a Vespa (Vega secondary payload adapter) and deorbit it [25]. Notably, as far as the author is aware, no dedicated ADR mission has removed a single piece of space debris.

Studies into active debris removal consider different targets. The selection of targets can be done in various ways as reviewed by Bonnal et al [26]. As suggested by the 2011 paper by Liou [27], many consider the LEO region between 600-800 km, especially at high inclinations, to be the most critical, and look for targets in those areas. In the same paper, Liou discusses different types of objects and objects of note that have the highest probability of collision in the LEO region. Wiedemann et al looked at the catastrophic debris flux and combined it with object mass to derive a criticality score for a debris object [28]. Most methods like these achieve similar results, and many identify Envisat as one of the most critical objects. Furthermore, Bonnal emphasized the opportunity to use the Kosmos 3M upper stage as a benchmark case, as many such objects exist in the crowded regions, and appear high in many priority lists. ClearSpace-1 selected the Vespa target because many such targets also remain in similar orbits, and are relatively cooperative.

Once a target has been captured, it needs to be taken out of orbit or have its orbit raised, to reduce the probability of collision. In a capture-based architecture, which the current proposed missions aim to use, the chaser includes an attitude controller and working actuators. The chaser then provides the attitude control after capture. This is a nontrivial task. Depending on the capture architecture, the connection with the target might be rigid or flexible, and initial attitude rates might be far from zero, requiring detumbling. In case of an uncooperative target, model parameters such as inertia or centre of mass might be unknown, complicating the control further.

1.3. Post-capture attitude control introduction

To help solve the problems described above, this research investigates the problem of post-capture control of an active debris removal mission. To provide the relevant background for this research, in this section the general ADR attitude control preliminaries are discussed, followed by the requirements used. Next, related and prior works are discussed.

1.3.1. General ADR attitude control background

Controlling the attitude of the post-capture combined spacecraft is made difficult due to the non-linearity of the dynamics and kinematics. Furthermore, the requirements for the attitude control can vary greatly for different missions or even mission phases. While many methods exist to deal with these complexities, the focus in this research lies on the case where reinforcement learning is used to control the attitude. However, to grasp the broader context and to be able to compare with baseline methods, relevant background of non-reinforcement learning control theory is discussed in this section.

For spacecraft attitude control, a controller is used in a closed-loop set-up, where the spacecraft state is fed back into the controller. However, an active debris mission requires several key capabilities. These include, among others, launch, rendezvous with the target, approaching the target, stabilization (of tumbling targets), capture, and post-capture control [27]. Often, to realize these capabilities different controllers or control modes are used. For example, some spacecraft have a safe mode that can take effect in case of failures or other off-nominal conditions. Often in such a scenario the attitude controller brings the spacecraft to a power-positive attitude or tries to dampen the body rates [29].

Other types of attitude control modes can include for example sun acquisition/pointing, (precision) pointing, slew mode, or spinning control [30]. To further illustrate the way these modes relate, a relatively

simple example of these different modes is shown for the ITASAT cubesat in figure 1.3. The phases that lead up to pointing control often require increasingly stringent conditions to be met before they can be successfully used. This allows many spacecraft to revert back to a courser (and often simpler and more reliable) control mode if required. This is useful in case of an emergency or for example for reaction wheel desaturation. Meeting accurate and precise pointing requirements becomes difficult in particular by uncertainty in the model.

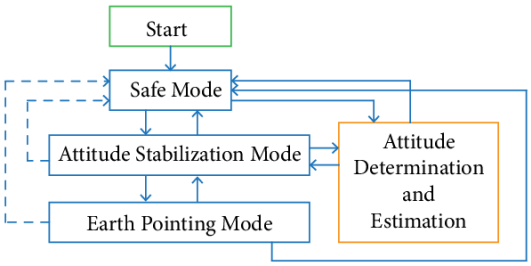


Figure 1.3: Attitude control modes for the ITASAT cubesat, from Carrara et al [31].

In the context of an ADR mission, immediately after capture, the spacecraft could be in a tumbling state, as is clear from the mission profile and corresponding requirements set by e.Deorbit and COMRADE[32][33]. Hence, a detumbling controller is required. While this is not a solved problem in active debris removal, relatively straightforward detumbling methods already exist. A good example of a method to achieve near-zero angular rates is the B-dot algorithm. This algorithm uses magnetic control, with a control law in which the magnetic moment produced by the magnetic actuators (often magnetic coils or torque rods) is proportional to the change in Earth's magnetic field. This method is not perfect, as it can result in observability singularities [34], or have trouble in fully arresting spacecraft rotation. Nevertheless, it is often used due to its relative simplicity. Furthermore, magnetic control also has applications beyond detumbling, for example in desaturation of reaction wheels.

After the detumbling phase of the ADR mission, the spacecraft needs to orient itself. It hence goes into a pointing mode. At the same time, in the case of ADR, the uncertainties are highest in the capture and post-capture phases. This is because the inertia or center of mass might be unknown or at least relatively uncertain. Moreover, if the system is flexible, uncertainties in the natural frequencies and damping of the flexible modes and the coupling strength could also increase the difficulty of meeting the requirements on attitude control. Hence, this pointing mode is the mode that is considered in this research. To do so, first, the requirements will be discussed. Then, existing solutions to this problem are introduced.

1.3.2. Requirements for post-capture attitude pointing control

The requirements on the attitude control of an ADR mission is highly dependent on the mission phase. For example, the capture phase requires higher pointing accuracy and lower angular rates, while for a spacecraft in safe mode a coarser pointing accuracy is often sufficient. The attitude control requirements of an ADR mission are very similar for spacecraft maintenance and servicing missions, which face similar problems [35]. For this reason, the pointing requirements of both the e.Deorbit mission and the COMRADE mission (designed to perform both ADR and servicing missions) will be used in establishing appropriate requirements for an ADR mission.

The e.Deorbit GNC performance requirements are taken from Telaar et al [32]. During the capture phase, the spacecraft's attitude shall be accurate to within 5° and the rate to within 0.5°/s. Post-capture, first, a stabilization (detumbling) phase is considered, which stabilizes the rotation to within 0.5°/s. Finally, the de-orbiting phase has requirements on the attitude and rates of 1° and 0.1°/s respectively during a boost maneuver (such as a thruster firing for de-orbit) or 5° and 0.5°/s respectively during a drift phase.

COMRADE considers only the phases up to stabilization. The requirements were taken from Colmenarejo et al [33]. They set the requirements of the attitude and attitude rate during the capture phase to be within 2° and 0.5°/s respectively, with a 95% probability. During the grappling phase, the

capture mechanism is restricted to within 2.74° in all three directions independently, and the rates to within 0.1°/s in all three directions. In the post-capture stabilization phase, Colmenarejo requires the rate to be stabilized to within 0.5°/s.

Table 1.1: Summary of the most stringent requirements from the e.Deorbit mission [32] and the COMRADE mission [33]. The post-capture phase requirements are relevant to this research.

Phase	Pointing accuracy requirement	Rate requirement
Capture phase	2°	0.5°/s
Post-capture phase	1°	0.1°/s

Attitude control systems are capable of achieving much finer attitude control. For example, many telescopes require a pointing to within less than an arcsec. Furthermore, in cases like these, vibrations and mean deviations should also be considered. In this case, the requirements can be formulated more specifically. For example, the Hubble Space Telescope can return to a target attitude to within 0.01 arcsec, but can hold an attitude to within 0.007 arcsec (root mean-square) for up to 24 hours [36]. These types of requirements will not be considered for this research, as they are driven by system-level capabilities (in case of Hubble, to allow near-diffraction limited imaging), which are not present on the considered ADR mission concepts such as those proposed by e.Deorbit and COMRADE. However, these might still be relevant to consider in future research. In general, for the post-capture phase, the requirements of the e.Deorbit mission are more difficult than those for COMRADE. Hence, those will be considered in this proposed research. The requirements are summarized in table 1.1. For the post-capture phase considered in this research, the relevant values are a pointing accuracy requirement of 1° and a rate requirement of 0.1°/s.

1.3.3. Non-reinforcement learning post-capture attitude control

As previously introduced, meeting these requirements is a non-trivial task given the non-linear dynamics and the uncertainties in the system. To simplify the analysis, some studies assume a rigid chaser spacecraft, a rigid target, and a rigid capture mechanism. The resulting combined spacecraft after capture is hence also often modeled as a single rigid body, for example in the work of Huang et al [37]. Other researchers, aiming for a higher fidelity model, have modeled the combined body as a flexible multi-body system [38]. While most studies are successful in implementing solutions that meet their requirements, this comes at a cost of more complex dynamics and kinematics, and a corresponding higher computational load. Models for the attitude control of combined spacecraft require controllers that are able to deal with non-linearities, and in the ADR case possibly with unknown or uncertain model parameters.

Previous research into this application and related topics can be roughly split into two broad categories: adaptive controllers and robust controllers. Adaptive controllers can adapt their model or control law online, for example by using system identification techniques. Robust controllers instead opt to design controllers that account for the uncertainty directly.

Both controller types have been thoroughly studied and applied abundantly in simulated ADR mission studies and concepts. Wang et al have shown an adaptive controller that can compensate for actuator faults, an error in the inertia tensor, and external disturbances [39]. Their implementation considers thrusters as the attitude actuators, without considering bounds on the thrust, or bounds on the control torque in general. Ning et al do consider this, and implement a fuzzy logic system. However, their case is for a cellular satellite system and they are not able to achieve asymptotic pointing accuracy [40]. Several proposed methods also use a baseline conventional controller which is augmented using neural net or general learning based approaches. In the context of ADR, this was demonstrated by Leeghim et al using a sliding mode controller with an adaptive term based on an normalized input neural network, with a five-neuron hidden layer [41]. Wei et al used a static prescribed performance controller supplemented by an approximate dynamic programming[1] controller to enhance adaptiveness also for the purpose of an ADR mission [42]. Both approaches achieve performance better than the

[1]The terminology of the class of methods referred to as reinforcement learning in this research is often interchanged in literature. Wei et al, and most control theory researchers, call this approximate dynamic programming. In this research, the name reinforcement learning is used.

baseline without the adaptive neural supplement, which indicates that learning-based can contribute to dealing with uncertainty at least in some cases. However, while both studies prove uniform ultimate boundedness of the signals, an assessment of the reliability and robustness of the controllers has not been carried out.

Robust control methods have also been studied extensively, such as multiple different techniques for robustifying a controller for a flexible ADR mission by Singh [38]. Their worked showed that while the controllers are able to achieve convergence in rigid body motion, certain eigenmodes of the flexible body are excited which the unfiltered controllers fail to appropriately dampen. They are able to dampen these out with a notch filter, notably using higher order filters, but they also note that this approach has limits due to computational constraints in tuning these filters. An example of their controller's performance, using a model of Envisat with a flexible solar panel, in the form of angular rates using are shown in figure 1.4. While their controller did not achieve arbitrarily small errors, the requirements discussed above could possibly be met with follow-on work.

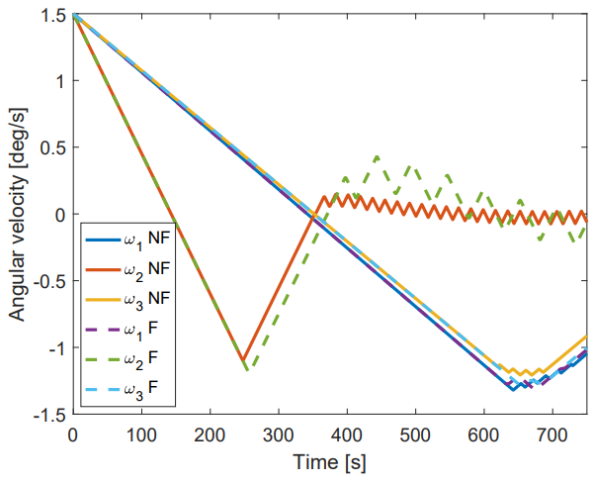


Figure 1.4: Angular rates for the flexible multibody INDI controller as designed by Singh and Mooij [38]. Note the remaining lower frequency vibration in the filtered (F) case, indicating a need for further filtering.

Huang et al used prescribed performance methods [43] to control a spacecraft while considering disturbances, inertia uncertainties, and actuator saturation. Luo et al and Huang and Duan also implemented variants of prescribed performance for similar cases [44][45]. A drawback of prescribed performance is that it is better suited for cases where the transient performance is important. It does not generally allow for asymptotic convergence to the target, and does generally not ensure limits on the amount of jitter, which can sometimes be important for spacecraft pointing. Furthermore, as noted in the former, designing a pure prescribed performance controller yields poor performance [43], hence, all three of the mentioned studies include disturbance observers. Such observers are often difficult to tune. Sliding mode and backstepping methods have also been applied to design robust controllers [46][47][48]. For each of these methods, complexities or drawbacks remain. For example, sliding mode controllers need a way to deal with the problem of chattering. Gao et al designed a sliding mode controller without chattering issues and capable of dealing with disturbances [46], an example of which has been shown in figure 1.5. However, their method is data-driven and requires a different controller in the first 100 seconds (in their implementation) to train a forecasting neural network. So, despite demonstrating the good performance visible in the figure, for a post-capture setting, this method cannot be used end-to-end.

The non-linear nature of satellite attitude control is also a key consideration in the design of a controller. One way to deal with this is the concept of non-linear dynamic inversion [49]. A drawback of this method is that it requires knowledge of or accurate identification of the system dynamics, which can be difficult, especially in the context of an ADR mission due to the uncooperativeness of some targets. Incremental non-linear dynamic inversion (INDI) [50] is a method which implements the dynamic inversion to an incremental model of the system. This relies on the assumption of a high sampling rate for the system

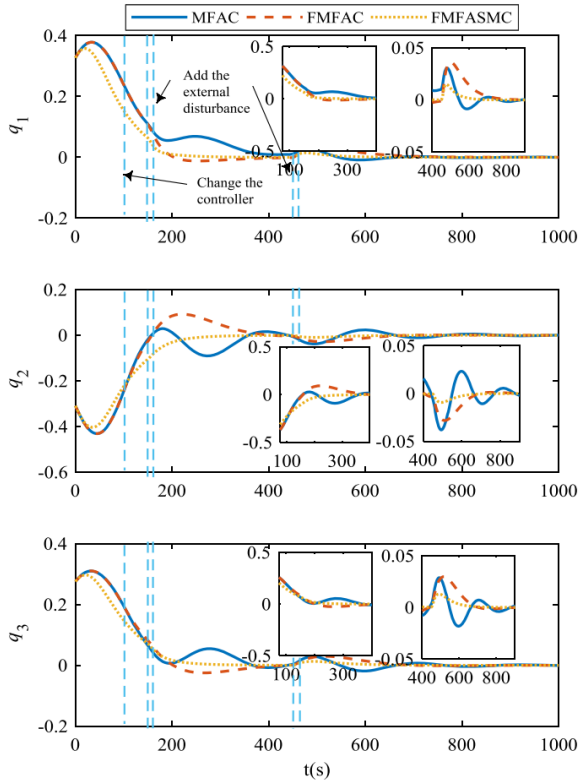


Figure 1.5: Vector components of the attitude error quaternion over time of a spacecraft attitude model by Gao et al [46] using three different attitude controllers.

states, and allows for an even higher degree of simplification in case of sufficient time scale separation. INDI has been applied in the context of spacecraft before, such as by Acquatella et al [51]. While their results are successful, they argued for further research into the real-world effects of actuator output and angular acceleration measurements. Further studies have made progress in designing a more robust controller using INDI [52][53], but the method has not been thoroughly tested yet for the case of ADR or takeover control with all its uncertainties. Singh and Mooij [38] did implement it for an ADR mission and achieved convergent behavior. However, as discussed previously, the flexibility of the satellite under investigation caused excitation of certain eigenmodes that caused the INDI controller to diverge slowly, which caused them to conclude that INDI was not sufficient for control of flexible perturbations.

Zhou et al investigated the case of a partial observability, using the example of spacecraft attitude control with fluid sloshing, and applied a policy gradient reinforcement learning method. They used an incremental model in closing the update loop for the policy weights [54], resulting in a model-based architecture. It still relies on accurate identification of this incremental model, which makes it vulnerable to sensor errors. Nevertheless, their results show that such architectures are able to converge relatively quickly, and that reinforcement learning methods can also be used in an end-to-end fashion, instead of having a conventional controller that is augmented by a neural-network based term, which is a relatively well-studied architecture both for spacecraft attitude control, such as Zhou et al [55] and other control problems with many nonlinearities and uncertainties such as robotic manipulator systems, such as Vo et al [56]. This greatly simplifies the design of these controllers, and is one of the reasons why an explorative study into the limits of these controllers is interesting.

1.4. Reinforcement Learning

Reinforcement learning (RL) is a form of machine learning that is based on the concept of maximizing a numerical reward signal [57]. This is very similar to a control problem, where the problem is often formulated in the form of minimizing some metric, often derived from the state error and control effort. Some reinforcement learning methods have advantages that make them compelling candidates for the

post-capture attitude control of ADR spacecraft. For example, a class of methods called model-free reinforcement learning methods do not require prior knowledge of a system model. Prior research such as that of Liu et al [58] claim to have demonstrated the application and robustness of such methods for an attitude controller for a disturbed rigid spacecraft. However, often, like in the case of Liu, it is not clear how certain design decisions have been made. These decisions, like the selection of feature basis functions or other hyperparameters, can influence the behavior and performance of the resulting policies dramatically. A comprehensive consideration of the various model-free RL algorithms, investigating the limits of their applicability, has not been performed.

Another interesting note is the fundamental difficulty of applying reinforcement learning algorithms in the real world [59]. While previous research has shown some success in real-world applications, for example in various robotic arm controller tasks [60][61], this is not without difficulty as many advances in reinforcement learning rely on assumptions that are rarely satisfied in real-world environments. Reinforcement learning methods depend on trial-and-error learning processes. Current algorithms require a large number of environment interactions to make meaningful progress and achieve sufficient generalization. This is called a high sample complexity. Training the algorithm using real-world interaction can be expensive or difficult to do to the required extent compared to training on a simulated environment. However, simulations will always involve mismatches with the real-world [62]. Thus, ensuring that the insights gained from the design of a reinforcement learning-based controller for a simulated model of spacecraft attitude control result in meaningful insights for the design of such a controller for a real spacecraft is a second priority for a research topic in this area.

1.4.1. Formal definition

A reinforcement learning algorithm is characterized and fully defined by a Markov Decision Process (MDP). An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. $\mathcal{S}$ is the set of possible states, and $\mathcal{A}$ is the set of actions, both of which can be continuous or discrete. Probability $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the probability that any state-action pair $s_t \in \mathcal{S}$ and , $a_t \in \mathcal{A}$ will result in state $s_{t+1} \in \mathcal{S}$ with probability $p(s_{t+1} \mid s_t, a_t)$. The reward function is specified by $r(s_t, a_t)$, which maps $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The factor $\gamma \in [0, 1]$ is called the discount factor. At $\gamma = 1$, the MDP is called undiscounted. A key factor in reinforcement learning is the (discounted) sum of future rewards:

$$J = \sum_{i=0}^{\infty} \gamma^i r(s_i, a_i) \tag{1.1}$$

A subset of all MDPs are the episodic problems. In episodic problems, there exists a set of terminal states $\mathcal{S}_T$, after which all subsequent rewards are zero, ending a so-called episode. To further interact with the environment, a new episode has to be initiated. In the episodic case, equation (1.1) becomes $J = \sum_{i=0}^{k} \gamma^i r$ where the terminal state is reached at step k. The other possibility is the continuing case, where equation (1.1) holds and is summed until infinity in the limit.

Reinforcement learning algorithms aim to maximize an objective function, generally the reward to go in equation (1.1), as will be shown here. The notation used is that of OpenAI's Spinning Up [63]. This is done by learning a policy $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$. The policy $\pi(a|s)$ determines the probability of choosing action a given state s. The optimal policy π^* is defined by the policy that maximizes J:

$$\pi^* = \arg\max_{\pi} J_\pi = \arg\max_{\pi} \sum_{i=0}^{\infty} \gamma^i r_\pi \tag{1.2}$$

where $r_\pi = r_\pi(a, s)$ are the rewards obtained by always selecting action a based on state s following policy $\pi(\cdot|s)$. For a given policy π, a state can be assigned a value using the value function, which is defined as the expectation of the reward to go under policy π when starting from state s:

$$V_\pi(s) \doteq \mathop{\mathbb{E}}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_0 = s \right] \tag{1.3}$$

A key insight first had by Bellman in 1954 [64] is that the value function can be defined recursively:

$$V_\pi(s) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \;\middle|\; s_0 = s\right] \tag{1.4}$$

$$= \mathop{\mathbb{E}}_{a\sim\pi}\left[r(s,a) + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \;\middle|\; s_0 = s\right] \tag{1.5}$$

$$= \mathop{\mathbb{E}}_{\substack{a\sim\pi\\ s'\sim p}}\left[r(s,a) + \gamma V_\pi(s')\right] \tag{1.6}$$

Where $s' \sim p(\cdot|s,a)$. Equation (1.6) is known as the Bellman equation. The optimal policy is also the policy that selects (possibly one of the multiple) actions that maximize $V(s)$. As a result, also the optimal value function can be defined as the value function of the optimal policy:

$$V^*(s) = \max_\pi \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \;\middle|\; s_0 = s\right] \tag{1.7}$$

or recursively as

$$V^*(s) = \max_a \mathop{\mathbb{E}}_{s'\sim p}\left[r(s,a) + \gamma V^*(s')\right] \tag{1.8}$$

So in turn, the optimal policy can also be defined as $\pi^* = \arg\max_\pi V_\pi(s)$. An important concept similar to the value function is that of the state-action value function. It is the expectation of the discounted rewards to go starting in state s, taking action a, and consequently following π. The action-value function can be defined respectively in terms of discounted reward to go or recursively:

$$Q_\pi(s,a) \doteq \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \;\middle|\; s_0 = s, a_0 = a\right] \tag{1.9}$$

$$= \mathop{\mathbb{E}}_{s'\sim p}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{\pi}\left[Q_\pi(s',a')\right]\right] \tag{1.10}$$

As such, the value function and action-value function are related as follows:

$$V_\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}\left[Q_\pi(s,a)\right] = \sum_{a\in\mathcal{A}}\pi(a|s)Q_\pi(s,a) \tag{1.11}$$

$$Q_\pi(s,a) = \sum_{s\in\mathcal{S}} p(s' \mid s,a)\big[R(s,a,s') + \gamma V_\pi(s')\big] \tag{1.12}$$

In equation (1.12), $R(s,a,s')$ is the reward function, defined for the more general case where the reward depends not only on the previous state-action pair (s,a) but also on the next state the MDP arrives on s'. Usually, this reduces down to $R(s,a,s') = r(s,a)$. The optimal functions also relate as follows:

$$V^*(s) = \max_a Q^*(s,a) \tag{1.13}$$

and the optimal action (and thus the optimal policy) can be taken from the optimal action-value function, which is very easy in the tabular case:

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{1.14}$$

To maximize the objective function in the MDP, different reinforcement learning algorithms employ different strategies. Bellman defined a set of techniques called dynamic programming [65]. In general, methods based on this technique sweep through the state space, performing some update on a state. Howard in 1960 proposed a process of alternating policy evaluation (calculating the value function for a policy) and policy iteration (improving the policy using the current value function), until convergence to the optimal policy value function [66]. Another major technique is the set of Monte Carlo methods, which update value functions based on environment interactions in the form of many complete episodes. An important implication is that generally, Monte Carlo methods do not need bootstrapping. Temporal

Difference learning is a technique that combines the bootstrapping and online learning potential of dynamic programming with the idea of learning from experience in environment interaction, like Monte Carlo methods [67].

Built upon fundamental learning principles like these, roughly two major classes can be identified: model-based methods and model-free methods. The basic information flow for a reinforcement learning agent is shown in figure 1.6. A model-based method learns a model of the system and uses this

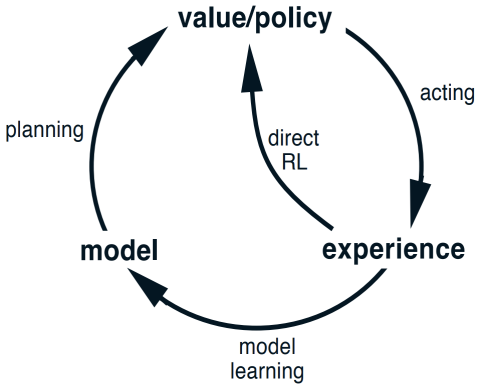


Figure 1.6: Schematic illustration of information flow in a model-based reinforcement learning agent, from Sutton and Barto [57]. In a model-free agent, no model is learned, and hence no planning is done. In that case, only the loop between the value/policy and the experience remains, and the value/policy is learned directly.

model in a planning stage. The aforementioned conventional dynamic programming requires a model of the system. On the contrary, a model-free method is essentially a trial-and-error based approach. Monte Carlo is a model-free approach, since it uses information sampled from real experience.

A second key classification set is the distinction between on-policy and off-policy learning. In reinforcement learning, there is always a balance between exploitation (the greedy policy; selecting the best action possible for the current state) and exploration (choosing a different action to find potentially better policies). This gives rise to a problem: the optimal policy is greedy with respect to the value function, and thus cannot do any exploration. If an algorithm uses only data sampled using the current policy to learn, it is an on-policy algorithm. If it is able to use experience sampled using a different policy (for example, a policy that explores more), it is an off-policy algorithm.

The third and final classification is the state and action space type. These may be continuous or discrete, which has a substantial impact on stability and convergence guarantees of algorithms. Many algorithms are only guaranteed to converge to a local or global optimum in the tabular (discrete) case, or sometimes only in the linear case [68]. Furthermore, discrete domains have a large risk of suffering from the curse of dimensionality [69], where computational costs rise exponentially due to the size of the search or state space $\mathcal{S}$. Function approximation, often by switching to a continuous domain, is a way of avoiding this issue.

Different algorithms trade off the pros and cons of these classifications, and might perform wildly different in different problems. Sometimes, it is possible to infer and predict how different algorithms will behave, but the results can also be surprising. To determine which algorithm is best suitable to a given problem, an in-depth analysis could be done.

1.4.2. Reinforcement learning-based post-capture attitude control

Before formulating an approach for designing a controller for the presented case of post-capture attitude control for an active debris removal spacecraft, the difficulties associated with such a problem need to be well understood, which is the topic of this section. Within the context of reinforcement learning for such an application, three focus points arise.

The first is the uncertainty in model parameters. For many target debris objects, the total mass, inertia, and center of mass will be unknown or uncertain [6]. As previously discussed, methods to estimate these already exist. However, the uncertainty in these dynamical parameters will never go to zero

and these methods can have other drawbacks. This means that the attitude controller needs to be able to adapt to unknown or uncertain model parameters post-capture, or be robust[2] enough to deal with this uncertainty directly. Simultaneously, the uncertainty in model parameters makes an agent difficult to train. Even the highest sample efficient reinforcement learning algorithms require a significant number of environment interactions to result in good policies, as RL algorithms are considered sample inefficient [70], which results in slow learning. Although research into speeding up learning has shown promise in the form of distributed reinforcement learning like Seed-RL [71], this is based on simulated experience instead of real experience, making training from scratch in the post-capture phase at least currently likely not feasible. Model-based RL algorithms that include a component of online system identification can significantly improve the sample efficiency, which enables a fast learning agent such as demonstrated by Zhou et al [72]. However, a value function approximation and policy still have to be trained off-line, while the system identification approach requires a set up assumptions to be valid (in their case, high-frequency data sampling and a relatively slow-varying system) which limits the applicability of the controller. As such, with current techniques, an RL-based agent used for a spacecraft attitude control purposes will likely have to be pre-trained before capture of the target.

The second difficulty is in training the agent for the real-world, as it has to be trained on representative samples. Training an agent in a real space environment is too expensive. Accurately setting up a simulated environment on the ground is difficult and also expensive. Hence, training has to rely at least partially on simulations, which will always be only an approximation of real world conditions. This type of problem is called sim to real transfer. A secondary effect of real-world effects, which includes considerations like actuator constraints, noise and bias in multiple system components, imperfect actuator responses, and signal and input delays, is that there is a risk of an apparent non-Markovian system for the agents. Reinforcement learning algorithms can have a hard time solving these and it can break performance and convergence guarantees of algorithms [73]. The non-Markovian can be caused by a model of too low fidelity. If the system is only partially observable, such as in case of a flexible spacecraft, it can also create the illusion of a non-Markovian environment [74], called a Partially Observable Markov Decision Process (POMDP).

Finally, the third problem is the required reliability of the controller. In space applications, reliability requirements are very stringent. The reliability of an attitude determination and control system can often be 99% [75] or more. Furthermore, depending on the mission, precise requirements on the pointing accuracy could exist. The problem with deep reinforcement learning is that the behaviour of learned policies are often difficult to explain, which makes the decisions taken using the policies difficult to verify [76]. Also, guarantees on stability of convergence can be difficult to obtain, be limited to guarantees such as uniformly ultimate boundedness [77], or even be impossible to obtain [78].

In dealing with the above issues directly, several influential and state-of-the-art algorithms are selected and discussed in section 1.4.3. In that section, general considerations of the discussed algorithms are highlighted. Only model-free algorithms are considered. Model-free methods are considered to suffer from high sample complexity [79]. However, since training on the ground is assumed to be required regardless of the RL algorithm type as discussed before, the computation time and hence the required sample complexity is not an issue. Furthermore, model-based methods require identification of the model, which can be computationally prohibitively expensive or difficult to achieve with sufficient accuracy. Methods to reduce the computational cost and reduce the difficulty of obtaining a sufficient fidelity model for model-based algorithms exist, such as shown by Zhou et al [54], who used an incremental model-based reinforcement learning approach, called incremental approximate dynamic programming. However, model-free methods have also shown promise in real-world applications, demonstrating their ability to perform well even without an approximation of the system in the feedback loop. An example is the application in dexterous in-hand object manipulation, which, even when trained only on simulated data, shows that model-free agents are capable of sufficient generalization for real-world applications [80]. Hence, model-based methods will be left for future research, while the focus is put on model-free methods in this research.

An influential idea in model-free control methods is the concept of Q-learning [81], where action-values

[2] In this literature review, the robustness of the controller refers to the definition of robust in the control theory convention (dealing explicitly with model uncertainty), not in the reinforcement learning sense (where robustness refers to an algorithm's ability to result in a good policy with low sensitivity to hyperparameter settings).

are learned. Given a state, the optimal policy selects the action based on the largest subsequent value of Q (equation (1.14)). Using policy evaluation, given sufficient samples and exploration, the optimal action-values can be calculated. This method does not require knowledge of the model (model-free), is off-policy, and is applicable for discrete state and action spaces. Mnih et al extended this idea to the function approximation case in highly influential works, showing excellent performance in various (simulated gaming) environments [82][83]. This approach still requires a solution for sampling an action from the Q-network. If the action space is discrete and relatively low-dimensional such as in the works of Mnih, this can be reasonably done by searching over the entire action space. If the action space is continuous, this becomes more difficult [84]. For example, Wang et al did this for spacecraft attitude control by discretizing the action space [85]. In case the continuous action space cannot be discretized, other constraints might be required to sample an action from the Q-network efficiently. For example, a linear combination of basis functions can be used as the action-value function approximation. This has been applied to the problem of spacecraft attitude control by Liu et al [58]. While they demonstrated performance better than an advanced sliding mode controller in an example simulation, they did not show robustness to different dynamical model parameters, nor did they share how they determined the appropriate feature vector for the linear Q-function, which is a critical consideration in ensuring appropriate generalization [57] and in the design of these agents.

For higher degrees of generalization, policy gradient methods have shown promise [86][84]. Instead of deriving a policy from (an approximation of) a value-function or action-value function, the policy is parameterized directly. This has two main advantages: the parameterized approximate policy can approach a deterministic policy over time if that policy is the more optimal policy, inherently trading exploration for exploitation, and the approximate policy can be stochastic, whereas action-value methods cannot result in stochastic policies naturally [57].

1.4.3. Policy gradient algorithms

Policy gradient methods that estimate both a value function (critic) and a policy (actor) are called actor-critic methods. These are the methods that have recently showed impressive results for multiple different applications, both simulated [84] as well as real-life [87].

Such actor-critic agents can be applied as a closed-loop controller. A generic architecture is shown in figure 1.7. The blue box highlights the boundaries of the RL agent, and represents the controller. As shown here, the environment also outputs some reward, based on the current state and latest action. The critic calculates a state (or sometimes action) value, which is combined with the reward information to calculate an advantage. This is used to calculate a gradient of the actor, and the value function itself is used to calculate a gradient of the critic, updating both neural nets.

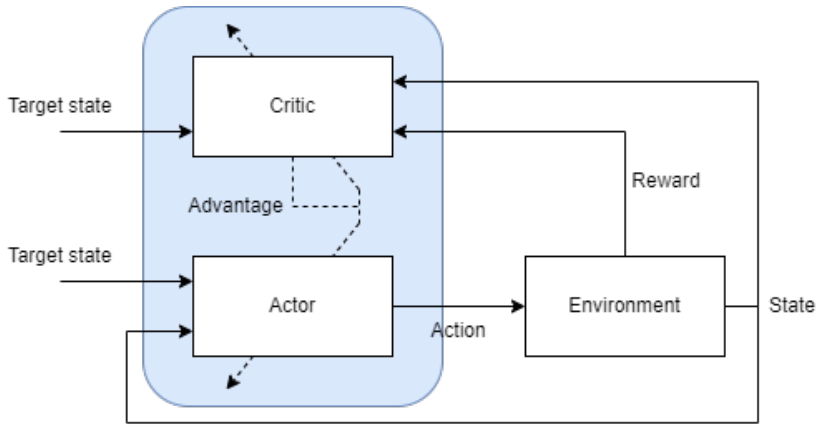


Figure 1.7: Generic actor-critic agent block diagram. Continuous lines indicate the flow of a signal, while the dotted line indicates the update rules for the actor and critic neural nets. The blue box indicates the scope of what is called the reinforcement learning agent.

Many variations on this block diagram exist, whether due to architectural design choices or due to real world effects. The neural nets might use different inputs, the update rules might be different, various

noise, disturbance, or delay components can enter in the block diagram, and the state might not be fully observable. Furthermore, the agent could save interaction tuples in a replay buffer to use for later training. In off-policy algorithms, a replay buffer is required, while some on-policy algorithms don't have one and have to sample new interactions after every update of the actor.

An influential example of an on-policy algorithm is Proximal Policy Optimization (PPO), introduced by Schulman et al [88]. It is both an improvement on and simplification of prior work by Schulman, the Trust Region Policy Optimization (TRPO) [89]. It is one of the most widely used on-policy algorithms. TRPO and PPO are both based on the concept of limiting the distance between the policy parameters before and after a policy gradient update. Both algorithms consider the KL distance between the policies as a constraint on the optimization problem. TRPO solves this directly using an algorithm such as conjugate gradient, while PPO (or specifically, as used in this research, the PPO-clip variant) solves it as a first-order method while clipping the objective. PPO has demonstrated great results in certain areas, such as in training a large language model like InstructGPT using humans in the loop [90], or more relevant, in the attitude control of a fixed-wing UAV [91] and even for spacecraft attitude control by Elkins et al [92][93]. In their first paper on using PPO, they showed that the trained agents can reliably point the spacecraft to within 0.005°, under simplifying assumptions such as no disturbances. They used a discretized action space, restricting the agent's commanded control output [92]. In their second paper, they switched to a (stochastic) continuous output of PPO, requiring sampling to obtain the output action of the agent [93]. In this work, they are able to slew the spacecraft using a PPO agent. In general, the drawbacks of an on-policy algorithm are the trade-off between exploration and exploitation and the required amount of episodes to train a policy. However, generally, on-policy algorithms are more stable [94].

Deep Deterministic Policy Gradient (DDPG) is an influential example of an off-policy algorithm. Introduced in 2015 by Lillicrap et al [95], it extends the deep Q-network method (DQN) by Mnih et al [82][83] to the continuous, deterministic control domain. It makes use of two (feedforward) neural networks. One to estimate Q (the critic), and one for the policy (actor). It is an off-policy method, sampling training data from a replay buffer.

Twin Delayed Deep Deterministic Policy Gradient (TD3), introduced by Fujimoto et al [96] improves upon DDPG in three significant ways. It reduces overestimations of the action-value function by adding a second critic, as first proposed by Van Hasselt et al [97]. Secondly, the variance and stability is improved by making use of target networks. Thirdly, it adds a small amount of clipped Gaussian noise to the target policy, which is a smoothing regularization to the policy. This reduces the potential for the policy to exploit narrow peaks in the value function. TD3 has been used in the context of spacecraft attitude control by Elkins et al [98]. In their later work, they compared it to PPO, and investigated methods for online learning using both agents. Some of their results are shown in figure 1.8. Across all scenarios, which is also visible in this figure, TD3 showed better performance. Furthermore, the stochastic sampled actions of PPO show up in a slight oscillation of the angular rates. They state the belief that the deterministic TD3 agent is hence better for such an application. However, it is unclear how the chosen settings like hyperparameters and the choice of their model influenced this comparison. Hence, a conclusion like this is too strong for the generalized case and merits further investigation, especially since the demonstrated performance is relatively good.

Recently, Fujimoto et al proposed a further improved algorithm that they named TD7 [99]. It adds four more improvements to TD3. It samples minibatch using LAP [100], a form of prioritized experience replay. In a fully offline learning case, the agent adds a behavior cloning term, which is only beneficial in a fully offline case where only a fixed dataset of environment interactions are possible. In this case it prevents divergence due to overestimations in the value function. The third addition is the inclusion of a representation learning method in the form of state-action learned embeddings. Representation learning has shown promise in tasks such as image classification or generative models such as GPT [101]. The implementation is done by adding two extra neural nets (encoders), one for the state and one for the state-action pair, which are trained in an unsupervised way and are aiming to capture relevant structure in the observation space. The resulting embeddings are concatenated to the features for the actor and the critic. The final improvement comes in the form of policy checkpoints, which in essence save the best performing policy during training.

The previously discussed policies have all been deterministic. The Soft Actor-Critic algorithm [102]

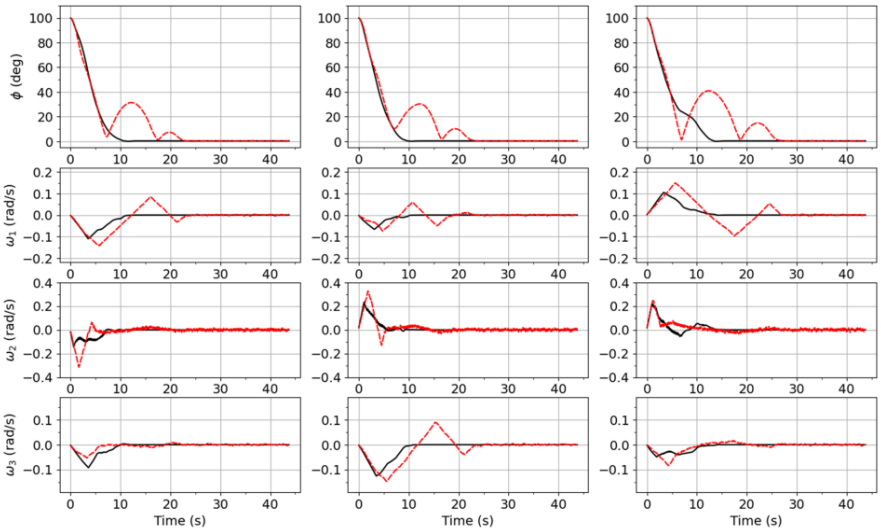


Figure 1.8: Euler axis rotation angle and attitude rates for TD3 (black) and PPO (red) agents, from Elkins et al [93].

augments the cost function of equation (1.1) to include an entropy term, scaled by a temperature hyperparameter, and models a stochastic policy. It is an off-policy algorithm. Due to the stochastic nature of the policy, exploration is inherent to the policy and is even encouraged (the policy is being optimized for future rewards while being as random as possible). The policy is able to reduce its stochasticity to become more deterministic if that is more optimal, which directly resolves the exploration versus exploitation issue. Also, the policy is able to capture multiple near-optimal behaviors [103].

TD7 is an example of a very recent, state-of-the-art algorithm. The authors claim performance equal to or exceeding the performance of other algorithms in a variety of benchmarks. Due to the novelty, this has not been independently replicated by other researchers. Soft actor-critic and PPO are often used in benchmarks for new algorithms, so could be considered state-of-the-art in that respect as well. Other examples of newer algorithms or improvements to existing algorithms do exist, such as TQN [104]. Current major areas of research include for example distributional algorithms such as Distributional Soft Actor-Critic [105], meta-reinforcement learning where recurrent neural networks are used [106], or using memory in reinforcement learning [107].

In this research, four algorithms were investigated, with the goal of identifying which type of algorithms are suitable for application in ADR post-capture attitude control. PPO was chosen to investigate a well-understood on-policy algorithm. TD3 is representative of a well-understood off-policy algorithm. SAC is also off-policy, but is stochastic instead of deterministic. Finally, TD7 represents a state-of-the-art algorithm. With these four algorithms, the research has a broad representation of the possible reinforcement learning algorithm, while being manageable in the available time.

1.5. Research objective

In the previous sections, it was established that active debris removal likely a necessary intervention to ensure sustainable access to space [13]. A likely candidate method for this, using space manipulators, is not without remaining challenges. A good example of such an open challenge is the post-capture attitude control of a combined chaser-target spacecraft.

Post-capture attitude control for an ADR mission is a control problem that has many uncertainties. The problem includes not just uncertainty in dynamical parameters such as inertia, but even in the way the problem can be approximated, such as the consideration of a rigid or a flexible body. Model free-reinforcement learning methods are known to in some cases exhibit properties that make them especially suitable for dealing with precisely these uncertainties, as they do not require knowledge of a model. Furthermore, they have been researched quite extensively and have shown promise in simulated scenarios as well as in some real-world applications. This is especially important in the case where offline training is possible before deployment in the real world, as is the case for ADR spacecraft

attitude control due to the costs of such tests.

While RL agents have been adopted for attitude control of a spacecraft before, a broad and comparative study of the merits of various algorithms in problems of this type has not yet been undertaken, which in the view of the author is a gap of interest in the literature. Executing such a study could result in two scientific advancements.

The first would be for the field of reinforcement learning. Studies comparing different algorithms for different use cases are numerous, and efforts have been made to make benchmarks representative of the real world [87][59]. However, it remains difficult to explain which aspects of an agent's RL architecture contribute to poor or great performance, which is also highly dependent on the environment. A comparative study for attitude control of even a non-ADR spacecraft is lacking, which complicates both the selection of existing algorithms as well as designing new algorithms for an ADR application.

The second advancement would be for the field of post-capture control. Post-capture control requires a high degree of adaptability or robustness from a controller, which reinforcement learning agents have been demonstrated to be capable of in the past in different problem settings. A study in this area could contribute to determining whether these capabilities of such agents could translate well to the application of post-capture control.

Apart from the argument of the two possible advancements mentioned above, a secondary consideration exists. Recently, the field of machine and reinforcement learning have shown rapid developments and impressive results. This forms the foundation for a scientific curiosity for the performance of these techniques in various applications, which makes it interesting to experiment with these RL methods on difficult problems, such as the problem of post-capture attitude control.

As a result of these considerations, the objective of the research is formalized as follows:

> To assess and compare the implementation and performance of state-of-the-art model-free reinforcement learning algorithms in the context of post-capture attitude control for an active debris removal mission accounting for dynamic and kinematic uncertainties.

This research objective is relevant due to the need for efficient, reliable, and autonomous control capabilities in active debris removal missions. With this goal, the results of the research will support the decision-making process in selecting a control algorithm for an ADR mission. Furthermore, it either inspires future research into reinforcement learning for post-capture control, or in cases of negative results, encourage research into alternative options. It also enables future research in the application of specific reinforcement learning algorithms in the field of spacecraft attitude control and contribute to a deeper understanding of reinforcement learning algorithms in various applications. Fundamentally, the major contribution of this research is to be a generalized reference for the design of RL-based attitude controllers in an ADR mission in the post-capture phase.

In the objective, dynamic and kinematic uncertainties are mentioned. The former includes uncertainties in model parameters such as inertial parameters and torque effectiveness. The latter includes assessing different state representations and sensor effects.

A final critical distinction in this objective is the inclusion of the model-free qualifier. This does not mean that the spacecraft is not modelled in the simulation process (the spacecraft modelling will be discussed in section 2.2), or that the uncertainties are applied in several different ways. It means that the algorithms are completely isolated from this model information, that the algorithms do not learn a model, and that the algorithms have to learn a policy directly from environment interaction experience, despite the applied uncertainties in the spacecraft model that make this experience less consistent.

The scope of the research was carefully set, based on the available literature and time constraints. Research opportunities for designing (RL) controllers for the capture phase are also highly relevant. However, it is deemed a good strategy to first assess the simpler post-capture phase, which in later research could be extended to the capture phase.

1.6. Research questions

The research objective is achieved by formulating an answer to the research question. The research question has been formulated as follows:

> To what extent can the implementation of model-free reinforcement learning algorithms impact the performance of post-capture control in active debris removal missions under the influence of inherent dynamic and kinematic uncertainties?

In this research question and in the rest of the research, the performance is measured and defined in relation to the requirements (see table 1.1). The performance is measured in several ways. These metrics are described in detail in chapter 2, specifically in section 2.4 and section 2.5. In short, the performance is acceptable or not based on an agents capability of pointing the spacecraft to the target attitude within the requirements (described in section 1.3.2) or not. The fraction of times an agent meets this requirement determines the convergence rate of the agent. Then, also the settling time, the total control effort, the total episodic reward, and the angular error at the final timestep are measured. Finally, the wall time for agent training (on the same hardware) is measured.

The research question is broken into four sub-questions (SQ), to provide more structure to the research process:

> SQ1: How can model-free reinforcement learning algorithms be effectively adapted for the design of a controller for post-capture control in the context of active debris removal missions?
>
> SQ2: To what extent is the stability and convergence of a model-free reinforcement learning-based controller reliable?
>
> SQ3: How do inherent uncertainties in the dynamic and kinematic properties of the system affect the performance of a model-free reinforcement learning-based controller in the post-capture phase of an active debris removal mission?

The sub-questions combined provide a comprehensive picture for answering the main research question. The sub-questions also nicely consider the different design phases. The first sub-question focuses fully on the algorithmic considerations of the research question, namely the effects of the architectural differences between the different algorithms. By implementing PPO, SAC, TD3, and TD7 and comparing them, it explores what the effects are of implementing different model-free reinforcement learning agents based on different principles such as on-policy and off-policy learning and stochastic and deterministic actors, and the effects of tuning the associated hyperparameters. The brittleness to hyperparameter settings are quantified, as well as the sensitivity to initial and final conditions. Finally, the required computational power is measured, which is compared relatively between different agents.

The second sub-question is fully quantitative. As previously explained, reliability is key for spacecraft attitude control. This sub-question aims to provide insights into what affects the reliability of the agents. The insights from sub-question 1 are used in this process. Controller performances is assessed for many episodes, statistically averaging the results and obtaining a quantified measure of the overall reliability of the agents. A theoretical study about or a proof of a bound on the reliability was not pursued, as the emphasis of the research question lies on the implementation of existing controllers instead. Hence, the research of convergence guarantees is outside the scope of this research.

The third sub-question extends the work from the prior two sub-questions by introducing the emphasis on the dynamic and kinematic uncertainties. This is done by testing the performance of the trained agents on many different spacecraft models with different dynamic and kinematic parameters. By doing so, a sense of the robustness of the controller is obtained. The effect of training agents with these uncertainties in the loop is also assessed.

The answer to the sub-questions are inter-dependent. The sub-questions were hence not sequentially

treated in both the research process and this report, but were rather taken as the reference upon which the methodology was set up in a comprehensive fashion.

The research questions together address one of two major promising characteristics of reinforcement learning algorithms: the ability to generalize, which possibly makes the agents more robust. The other major characteristic is the possibility to fine-tune trained agents, which possibly makes the agents more adaptable. This decision was made to maintain a manageable scope, so that a thorough investigation into the robustness was possible. Because of this, exploring the adaptability of the agents is the first recommendation for possible follow-on research.

1.7. Report structure

The report is structured to provide the reader a good understanding of both the individual components that make up the full model, as well as how the components are connected. This was started in this chapter, where the background to the research problem was given and discussed, and the research question was motivated and formulated.

Chapter 2 will discuss how the research questions are transformed into a methodology to answer them. This includes an overview of the general flow of information processing, modelling the spacecraft, and setting up the reinforcement learning agents. It also discusses specific settings in terms of hyperparameters and reward function formulation. In addition, the definitions for performance are formulated and measured. The chapter is closed by a discussion on what off-baseline settings the trained agents will be tested in to assess their robustness, and how training using domain randomization is used with the goal of improving the robustness.

Before the results are presented, first, the methodology is underwritten by the supporting verification and validation work that was done in chapter 3. This chapter starts with discussing the different unit tests and verification of generic software tools. Verification work of the custom PID and TD7 implementations are showcased, after which both the rigid and flexible spacecraft models are verified against other works. The full learning process is validated by comparing against the work of Elkins [92][98].

Chapter 4 next goes into detail about the obtained results. This is done in sections, building up in complexity. The research sub-questions are answered one-by-one, so that an answer to the full research question is formulated. This starts with a comparison between the baseline agents, where the four algorithms are compared against each other and against a reference PD controller. Then, the influence of the reward function on agent performance is discussed. Effects of changing hyperparameters, in particular including sensitivity to the hyperparameters, are presented next. This is followed by the results of the domain randomization experiments. Finally, the influence of flexibility in the spacecraft model is assessed in detail.

The report is concluded in chapter 5. The research question is answered, and the answer and the full work is evaluated. The core insights that result from this work are discussed. The chapter also includes recommendations for future work.

2

Methodology

To gain the insights that are required for sufficiently answering all the sub-questions comprehensively, a modular information processing pipeline, from here called the 'full model', was required. This full model needed to be able to easily switch between a rigid and flexible spacecraft model, including corresponding uncertainties, and be able to use several reinforcement learning algorithms in a learning process. It also needed to do this while retaining a common programming interface to run a large number of different experiments with relative ease.

The full model was implemented in Python. This choice was made due to the strong suite of public libraries available for reinforcement learning and machine learning in general. This would both aid in the implementation of the selected reinforcement learning algorithms, as well as the spacecraft model.

Before explaining the full model and its parts, some definitions are given. In this report, a simulation refers to a single training process of a single agent. From a higher level, an 'experiment' refers to a group of simulations with a common set of simulation settings (reward function, hyperparameters, model type and settings, etc.). The utility in repeating simulations with the same settings is to get the average result and a sense of the distribution of the results, which is stochastic due to the random initializations. On the highest level, a 'study' is referring to a group of experiments that investigate a similar effect. For example, a study could be optimizing a single or group of hyperparameters, training a model with noise inserted in the process, or another common variable to be studied. Finally, an 'environment' is referring to the spacecraft model. This includes the rigid and flexible spacecraft models, and all settings like inertia tensors, maximum control torque, etc. It should be noted that the definition for environment also contains the reward function, which is implemented as a wrapper around the spacecraft model.

A schematic overview of the full model is shown in figure 2.1. The process starts on the left, with information flowing to the right. All study settings are set in a study configuration file. This is where any setting can be changed that is relevant for the scope of the research, including agent selection, specifying hyperparameter values or hyperparameters to be automatically sampled, reward functions, or the domain randomization parameters, for example.

This chapter explains the thought behind the full model, and describes choices made in the process of setting up the model. It also details individual components. First, general notes on spacecraft control are described in section 2.1, and how those are taken into consideration in the full model. Second, the different spacecraft models are discussed in section 2.2, including both the rigid and flexible spacecraft models. Third, the setup of the reinforcement learning agents is discussed in section 2.3. This includes methodology for the problem formulation as well as the reward function, specific agent settings (including decisions made about the hyperparameter settings used as a baseline), and how the learning process is set up for each agent. The final two sections, section 2.4 and section 2.5 discuss how the learning process and the robustness of the experiments and trained agents are assessed, respectively. Combined, they provide a definition of performance, which is used to answer the sub-questions directly.

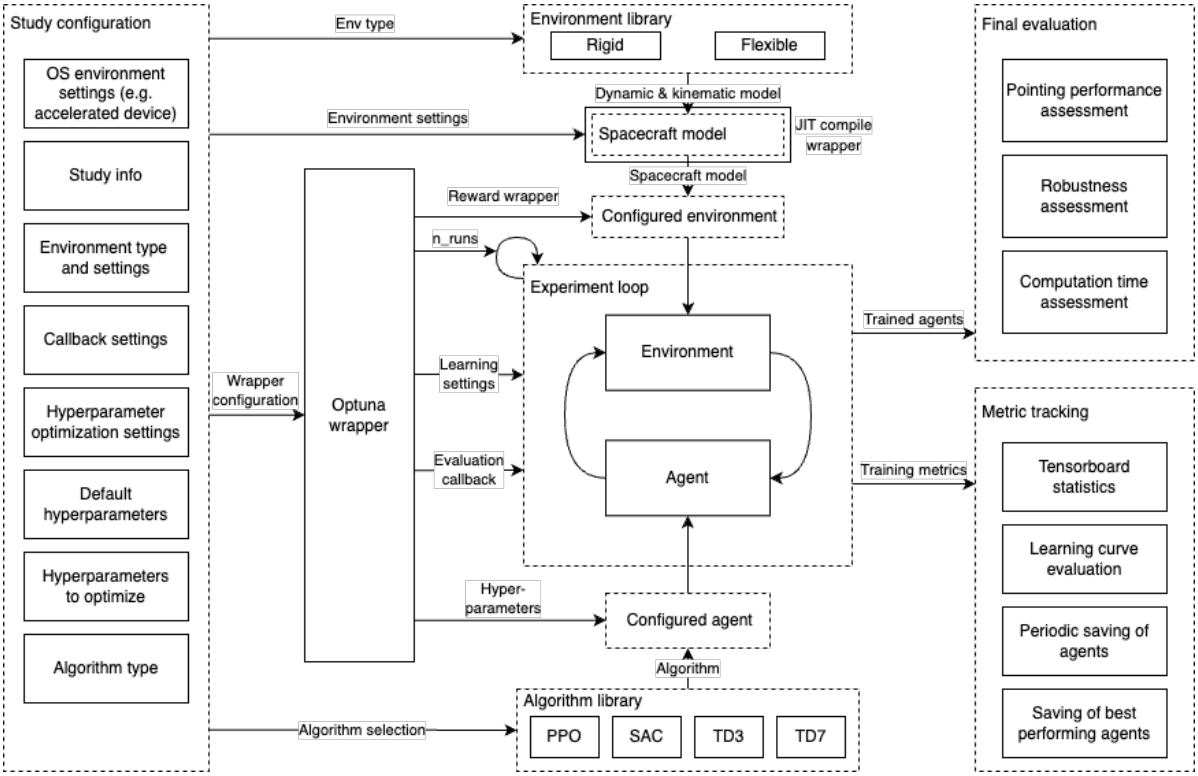


Figure 2.1: Schematic representation of the full model.

In general, the methodologies of this research can be categorized in two types of contributions. The first sections (until section 2.3) are largely taking various prior research results and existing methodologies and combining and adapting them appropriately for answering the research question. The second category is that of more novel contributions. The assessment method in section 2.4 is a novel contribution, as are the robustness assessment (including the sensitivity analysis) methods of section 2.5.

2.1. Attitude control simulation architecture

The process of controlling a spacecraft under the dynamic and kinematic uncertainties described in chapter 1 is a problem that is non-trivial, as there exist many real-world effects that are difficult to account for in a model. The fundamental control architecture that is used in this report is the closed-loop controller shown in the block diagram in figure 2.2, discussed below.

The attitude control is done using a closed-loop control system, where the controller (red) tries to track a reference attitude q_{ref} and attitude rate ω_{ref}. This reference is combined with the previous best estimate of the spacecraft's attitude and rates, which result in an error representation (q_e, ω_e). Based on this error, the controller outputs a commanded control torque $\boldsymbol{u}_c$, which is clipped to the control torque limits $(\boldsymbol{u})$, and is forwarded to the spacecraft model (in blue). An external disturbance torque $\boldsymbol{u}_d$, if applicable, is also taken into consideration, as well as various system uncertainties, such as thrust misalignment.

Based on the control and disturbance torques, the spacecraft model is then propagated further in time numerically. The choice of representation for the attitude and rates and the numerical integration setup can result in a buildup of an integration error. Furthermore, if the spacecraft model does not accurately represent reality, then a modelling error will also be present. All spacecraft models rely on certain assumptions for their validity. Central to this report is an assumption of a rigid or flexible spacecraft. However, even under this assumption, if uncertainties exist in this model such as uncertainties in the inertia tensor, these can highly influence the modelling error term. The numerical integration step is set to a value that limits the numerical error to negligible sizes. However, the modelling error is deliberately neglected in this research. As previously explained, this is due to the cost of a real-life comparison, and relevant real-life data for validation purposes is not readily available. The state that is outputted by

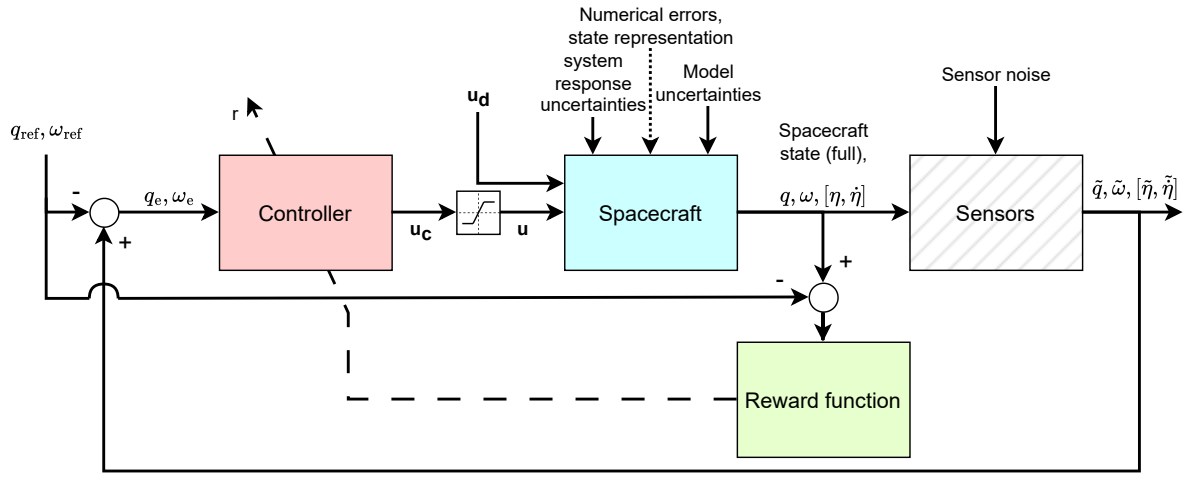


Figure 2.2: Full block diagram for the learning process for a spacecraft attitude reinforcement learning controller, color-coded: The blue box is the spacecraft model discussed in section 2.2, the red box is the (reinforcement learning) controller discussed in section 2.3.4, and the green box is the reward function discussed in section 2.3.2. The gray dashed box represents the sensors.

the spacecraft model is hence treated to be the true system state.

The system state is measured by the sensors (gray dashed). No sophisticated state estimation is considered, since that is not within the scope of the research. The sensors can introduce noise or other imperfections in the measurement of the state, but in the baseline case are considered ideal and perfect sensors. This measured state $\tilde{q}$, $\tilde{\omega}$ (and in the fully observable flexible case also modal coordinates $\tilde{\eta}$ and $\tilde{\dot{\eta}}$) is fed back into the controller.

Finally, the system state is used directly for the calculation of the instantaneous reward. This is used to train the controller. The system state is used directly, rather than the measured state. While it is possible to use rewards obtained from noisy measurements as shown by Wang et al [108], if not dealt with appropriately, it can be detrimental to learning performance, as Everitt et al showed [109]. This consideration, however, falls outside of the scope of the research, as the focus lies on the offline learning of the agents, which is a setting in which the spacecraft state is known without errors, improving the learning signal for the agent.

2.2. Spacecraft dynamics and kinematics model

Fundamentally, the objective is to train an RL agent to achieve the highest performance in a real-world spacecraft attitude control scenario. However, real-world experience for this specific case of post-capture control is not readily available to train the RL agent. Furthermore, what real-world experience exists is not easily modified to be representative of the wide range of uncertainties that will be investigated (as later discussed in section 2.5). Hence, a simulated spacecraft is used to generate the experience.

Because a simulated spacecraft is used, the insights gained from using the selected spacecraft model do no longer transfer 1 to 1 to the real world. This is exactly why the uncertainties are applied, so that an understanding can be obtained about the effects of particular types of real-world effects on the performance of these agents, without having to use a model with extremely high fidelity. Thus, two fundamental types of relatively simple models are used: a rigid spacecraft model and a flexible spacecraft model. These are described in this section.

The first three subsections describe respectively general target considerations, the reference frames used, and the control torque actuator and numerical settings. Then, the rigid spacecraft model is presented. Finally, a flexible spacecraft model, based on a modal coordinate representation of internal vibrations, is discussed. For these models, as well as sign, reference frame, or notation conventions, the book of Markley is used [110].

2.2.1. Target selection

Key to modelling the dynamics of the combined spacecraft is the inertia tensor, which describes something about the mass distribution of the spacecraft. In the active debris removal context of this research, as discussed in chapter 1, this combined spacecraft consists of a chaser and a target. Hence, first, a target needs to be selected.

As stated in section 1.2 based on literature [26][28] the most critical objects for ADR missions include Envisat, as well as a group of Kosmos 3M upper stages. For this research, it was decided to focus on Envisat, as many other ADR studies consider this object which makes it well-studied. Because it's well studied, its dynamical properties are well-characterized. Furthermore, selecting Envisat for this research, opportunities for future comparisons arise with similar studies, such as the extensive e.Deorbit study [19].

The chaser differs wildly between different studies. This among other factors depends on the objective of the specific ADR mission under consideration: what capture and removal method is used, will it capture one or multiple targets, and what is the target? The e.Deorbit chaser is significantly smaller than Envisat [19]. It is hence difficult to select an appropriate chaser (and corresponding dynamical parameters) without losing generality. Instead, it is more useful to assess the effects of changes in the inertia tensor on the behavior of a controller. For this reason, the chaser inertia is simply neglected, which is further motivated below. The inertia tensor that is used for this research is hence simply the inertia tensor of Envisat. The values for Envisat's inertia tensor have been taken from Arroz and Gandía's report [111]:

$$J = \begin{bmatrix} 17023.3 & 397.17 & -2171.4 \\ 397.1 & 124825.7 & 344.2 \\ -2171.4 & 344.2 & 129112.2 \end{bmatrix} \text{ [kgm}^2\text{]} \tag{2.1}$$

As described in the research question, the goal is to assess whether reinforcement learning algorithms has potential for "post-capture control in active debris removal missions under (…) dynamic and kinematic uncertainties" in a broad generalized sense, not for any specific mission. To achieve this, the inertia tensor is varied in two major ways. The first is a change in scale, which is implemented by scaling the components of the diagonalized inertia tensor. The second is a rotation of the inertia tensor.

These changes will result in a change in the rotational coupling behavior and time scale of the final dynamics. Provided the dynamics and associated uncertainties are representative of the active debris removal mission scope, these changes in behavior are also appropriately varied when simply only using Envisat's inertia and no chaser inertia. This means that the research question can still be answered appropriately. Note that a different chaser in the flexible case can influence other dynamic and kinematic factors as well, such as natural frequencies of internal vibrations, but this will be further discussed in section 2.2.5.

In conclusion, the chaser inertia is neglected, and Envisat's inertia is used as a baseline for all experiments. The effect of random rotations or random changes to the mass distribution is also studied, to ensure the effect of uncertainties and variations in inertia tensors are represented appropriately. It should be noted that these uncertainties are not necessarily present for Envisat's inertia tensor, but they are present for other potential ADR targets, and thus are still relevant to investigate. The specific implementation of these inertia modifications are discussed in more depth in section 2.5.

2.2.2. Reference frame

In order to put any description of a spacecraft in the right context, a key requirement is a well-defined reference frame of the model. Two reference frames are used in the context of this research: a body-fixed frame, and an inertial frame.

The body fixed frame B is attached to the spacecraft. For Envisat, the reference frame in figure 2.3 is used, based on Virgili et al [112]. This frame rotates with the spacecraft, with respect to an inertial frame.

The inertial frame, I is the second important reference frame. It is the frame in which Newton's laws of motion are valid, which form the basis for deriving further equations, such as Euler's equation equation (2.2). It is an unchanging and unmoving reference frame.

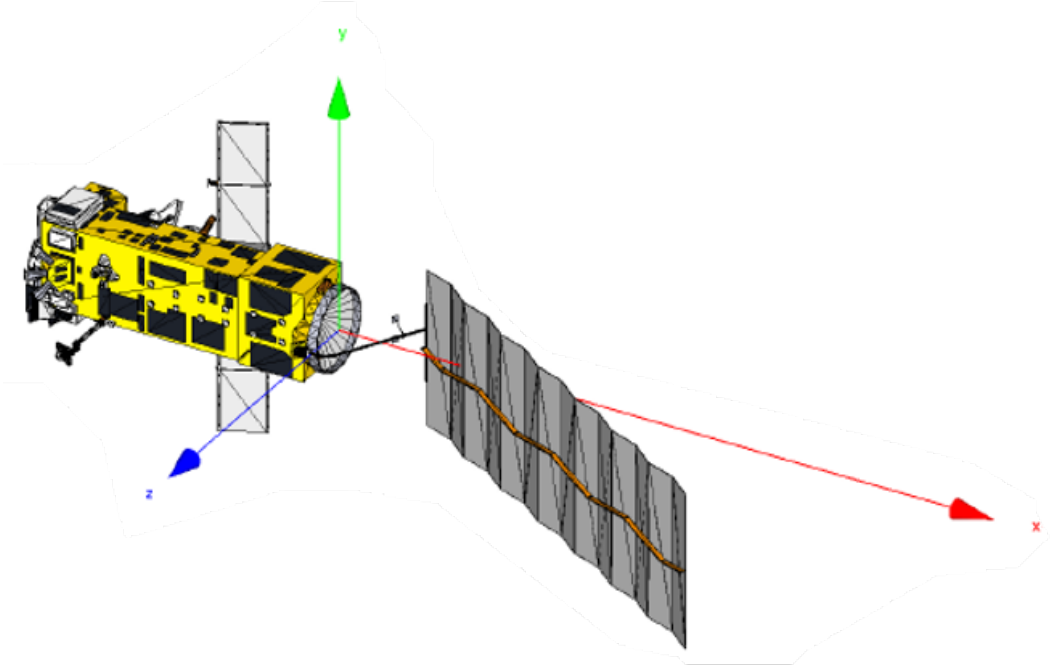


Figure 2.3: The Envisat body frame, from Virgili et al [112].

A third important reference frame in spacecraft attitude control is the local vertical local horizontal reference frame. It is often convenient to use this reference frame, for example to describe the attitude of the spacecraft of an Earth-pointing spacecraft [110]. The reference frame is defined in relation to the local horizon, the orbital angular momentum vector, and nadir direction. The difference with the inertial frame is expressed in terms of a coriolis term (under assumptions of a simple orbit). In this research, however, the orbital motion of the satellite is not directly considered. This enables a full focus on the attitude control, while removing the extra complexity introduced by the orbital motion. The local vertical local horizontal reference frame is thus not used further in this work. The assumption of an inertial frame is valid for investigating attitude control as the attitude dynamics of the system do not meaningfully change on a relevant timescale for ADR missions due to orbital motion effects.

2.2.3. Control torque actuator selection and numerical settings

The next step in setting up a spacecraft model requires the selection of an appropriate control torque actuator. In case of a mission design process, this selection can be driven by requirements such as a time limit on a slew manoeuvre. However, the research question again aims to answer the appropriateness of an RL controller for ADR post-capture attitude control in a general sense, without necessarily constraining to a single slew time. A fixed slew time, combined with a fixed inertia tensor, dictates the maximum control torque that the actuator needs to provide. Together, these also constrain the time scale of the dynamical system. As the RL agent will learn a policy in a simulated manner, a larger (or smaller) maximum control torque can be used in the simulation, as long as the simulated time (and controller sampling frequency) is also scaled appropriately when drawing conclusions from this research for a specific mission in the future. Hence, this design decision is also considered free.

The design decision for maximum control torque was taken in tandem with the sampling frequency. The sampling frequency was set to 1 Hz, based on a conservative estimate for the availability of sensor measurements. Initial experiments indicated that the learning speed of the reinforcement learning agents was influenced by the maximum control torque, dependent on the agent hyperparameters. The maximum control torque was hence set to a rough optimum of learning performance (based on the early trial experiments) of $u_{\text{max}} = 200$ Nm.

Finally, the numerical integration scheme and integration time step used are a Runge-Kutta 4 integration scheme with a timestep of 1/60 s. In the timescale of the system dynamics and the maximum integration

time, these settings proved sufficient to keep the numerical error negligible in comparison to other errors or uncertainties (as verified in section 3.1.3). This timestep size means that 60 propagation steps are done per controller sampling step. During these propagation steps, the control torque is kept constant. Changing this to semi-impulsive control, where only the first propagation step used a non-zero control torque provided by the controller, like the work of Elkins [93], did not affect learning performance in early tests and was not further considered.

2.2.4. Rigid spacecraft dynamics and kinematics

The underlying assumption for what in this report is called the rigid spacecraft model is, unsurprisingly, that the spacecraft is fully rigid. The implication of this assumption is that the dynamics of the system can be described fully by Newton-Euler dynamics. Following Markley's book [110]:

$$\dot{\boldsymbol{\omega}}_B^{BI} = (J_B^c)^{-1}\left[\boldsymbol{M}_B^c - \boldsymbol{\omega}_B^{BI} \times (J_B^c \boldsymbol{\omega}_B^{BI})\right] \tag{2.2}$$

Here, ω_B^{BI} represents the rotational rate of the body frame B with respect to the inertial frame I, in the body frame B. J_B^c represents the inertia tensor at the centroid, also expressed in the body frame B. Finally, M_B^c is the sum of the external torques on the body, acting at centroid c and expressed in the body frame B. The symbol $\times$ indicates the cross product operator. The reference frames are dropped for the rest of this report for ease of reading, and can be assumed to be the same as in this equation, unless otherwise clear from context.

In this definition, $\boldsymbol{M}_B^c$ is the sum of all external torques. This means that it both contains the commanded control torque $\boldsymbol{u}$ (modelled using $\tilde{\boldsymbol{u}}$, which includes its uncertainty effects such as misalignment, as described later in section 2.5) and possible disturbance torques $\boldsymbol{u}_d$ are summed and represented by this single term:

$$\boldsymbol{M}_B = \tilde{\boldsymbol{u}} + \boldsymbol{u}_d \tag{2.3}$$

It should be noted that the model in equation (2.2) does not assume a specific type of actuator. For example, in case of reaction or momentum wheels, additional coriolis terms should be accounted for due to the rotation of these wheels, or in case of a thruster-based system, the system mass (and hence inertia tensor) should also change due to the expelled propellant. These effects are neglected here, because the dynamical perturbations that these real actuators cause are of the same type that will be investigated by perturbing the inertia tensor and actuator response by the methods described in section 2.5.2.

To obtain the control torque $\boldsymbol{u} = \begin{bmatrix} u_1, u_2, u_3 \end{bmatrix}^T$, the commanded control torque $\boldsymbol{u}_c = \begin{bmatrix} u_{c,1} & u_{c,2} & u_{c,3} \end{bmatrix}^T$, which is the output of the controller, is clipped to a maximum control torque:

$$u_i = \begin{cases} u_{max} & \text{if } u_{c,i} > u_{max} \\ u_{c,i} & \text{if } u_{min} \leq u_{c,i} \leq u_{max} \\ u_{min} & \text{if } u_{c,i} < u_{min} \end{cases} \tag{2.4}$$

for $i = 1, 2, 3$.

The response of the spacecraft dynamics described in equation (2.2) to a given control torque are fully determined by the initial attitude rate ω_0 and the inertia tensor J. These terms are therefore analysed and set appropriately as described below.

An appropriate setting for an initial rate of the system is determined by the control phase the spacecraft is in, as discussed in section 1.3.1. As discussed, detumbling is relatively easy to do via other methods, which can be quite a bit cheaper. Furthermore, early tests indicated that training a policy that could detumble a spacecraft was relatively trivial. The interesting phase is the pointing control, where difficult requirements can exist on the pointing accuracy. For such a control phase, the initial rates are very small. As a result, all subsequent simulations were done with an initial rate of 0.

The decision to use 0 initial rates instead of a small rate was done so that a random factor contributing to the variance of the results could be eliminated. The assumption is that the results from a zero initial rate is not significantly different from one with a small non-zero rate, as the agents will encounter many states with non-zero rates during the learning process anyway. Furthermore, metrics such as the

settling time become a more reliable measure of the agent performance, due to the lower reliance on the initial conditions: with zero initial rates, it is only dependent on the initial attitude and policy itself.

The second part of the spacecraft model is the representation of the attitude. For this, several options exist. These options are listed with their pros and cons in table 2.1. Due to the linearity of the kinematic equations and the lack of a singularity, this research uses a quaternion representation. The added computational load due to the fourth parameter is insignificant in comparison with the computational load due to training the neural networks of the RL agents.

Table 2.1: Comparison of Attitude Representations

Attitude Representation	**Pros**	**Cons**
Direct Cosine Matrix	• Clear physical meaning • No singularities	• 9 parameters to track • Orthogonality constraint to be enforced
Euler angles (Tait-Bryan)	• 3 parameters to track • Intuitive understanding	• Singularities at pitch = $\pi/2$ • Non-linear derivative • Multiple angle conventions
Quaternions	• No singularities • Linear derivative	• 4 parameters to track • Not intuitive • Unit length constraint to be enforced
Modified Rodriguez Parameters	• Singularities can be avoided • 3 parameters to track	• Not intuitive • Non-linear derivative

The quaternion is defined in terms of euler axis $\boldsymbol{e}$ and euler angle ϕ as:

$$\boldsymbol{q}(\boldsymbol{e}, \phi) = \begin{bmatrix} \boldsymbol{e}\sin(\phi/2) \\ \cos(\phi/2) \end{bmatrix} \tag{2.5}$$

The kinematics of the rigid spacecraft are defined by [110]:

$$\dot{\boldsymbol{q}} = \frac{1}{2}\Xi(\boldsymbol{q})\boldsymbol{\omega} \tag{2.6}$$

where:

$$\Xi(\mathbf{q}) \equiv \begin{bmatrix} q_4 I_3 + [\mathbf{q}_{1:3}\times] \\ -\mathbf{q}_{1:3}^T \end{bmatrix} = \begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix} \tag{2.7}$$

This is dependent on the convention that $\boldsymbol{q}_{1:3}$ is the vector part of the quaternion and q_4 is the scalar part, which means that:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_{1:3} \\ q_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix} \tag{2.8}$$

After every numerical timestep (using RK4 as described in section 2.2.2), the quaternion is normalized to ensure the quaternion is always a versor.

The state error, computed between the measured states (which is computed on the left side of figure 2.2), is calculated by:

$$\boldsymbol{q}_e = \begin{bmatrix} q_{e,1} \\ q_{e,2} \\ q_{e,3} \\ q_{e,4} \end{bmatrix} = \tilde{\boldsymbol{q}} \otimes \boldsymbol{q}_{\text{ref}} = \begin{bmatrix} \tilde{q}_4 & \tilde{q}_3 & -\tilde{q}_2 & \tilde{q}_1 \\ -\tilde{q}_3 & \tilde{q}_4 & \tilde{q}_1 & \tilde{q}_2 \\ \tilde{q}_2 & -\tilde{q}_2 & \tilde{q}_4 & \tilde{q}_3 \\ -\tilde{q}_1 & -\tilde{q}_1 & -\tilde{q}_3 & \tilde{q}_4 \end{bmatrix} \cdot \begin{bmatrix} q_{\text{ref},1} \\ q_{\text{ref},2} \\ q_{\text{ref},3} \\ q_{\text{ref},4} \end{bmatrix} \tag{2.9}$$

$$\boldsymbol{\omega}_e = \begin{bmatrix} \omega_{e,1} \\ \omega_{e,2} \\ \omega_{e,3} \end{bmatrix} = \tilde{\boldsymbol{\omega}} - \boldsymbol{\omega}_{\text{ref}} \tag{2.10}$$

Here, the attitude $\tilde{\boldsymbol{q}}$ and $\tilde{\omega}$ are the quaternion attitude and attitude rates, respectively, as measured by the sensors in figure 2.2.

2.2.5. Flexible spacecraft dynamics and kinematics

The flexible model formulation used to investigate the research question is another non-trivial decision. This is because the model fidelity can be almost arbitrarily complex, using for example multi-physics numerical model. However, modelling the intricacies of the flexibilities encountered in the post-capture phase of a robotic active debris removal mission is not the focus of the research. A balance was sought between model fidelity and model simplicity, while keeping computational cost down.

A suitable model for this case has been used for a long time by NASA [113] and ESA [114] for relatively simple analysis of flexible models. Gennaro's adaptation [115][116] of this model serves as the basis for the implementation used in this research. This is because this model can capture flexible interactions of multiple flexible components, under different boundary conditions. It is based on a dimensionless modal coordinate representation of shape function amplitudes $\boldsymbol{\eta}$, following:

$$u_x(z,t) = \sum_{i=1}^{N} \Phi_{xi}(z)\eta_i(t) = \boldsymbol{\Phi}_x^T(z)\eta(t) \tag{2.11}$$

$$u_y(z,t) = \sum_{i=1}^{N} \Phi_{yi}(z)\eta_i(t) = \boldsymbol{\Phi}_y^T(z)\eta(t) \tag{2.12}$$

$$\theta_z(z,t) = \sum_{i=1}^{N} \Phi_{zi}(z)\eta_i(t) = \boldsymbol{\Phi}_z^T(z)\eta(t) \tag{2.13}$$

This describes the deflection $\boldsymbol{u}(z,t)$ of a point at coordinate z along a flexible beam with end mass m_p at time t, using a linear combination of u_x, u_y, and θ_z, following the geometry as shown in figure 2.4. N represents the number of modes considered, and Φ are admissible functions that depend on the boundary conditions.

The modal coordinates $\boldsymbol{\eta}$ are coupled with the attitude rates of the main body. Together, they are modelled as:

$$J_{mb}\boldsymbol{\omega} = -\boldsymbol{\omega} \times (J\boldsymbol{\omega} + \delta^T\dot{\boldsymbol{\eta}}) + \delta^T(K\boldsymbol{\eta} + C\dot{\boldsymbol{\eta}}) + \boldsymbol{M}_B \tag{2.14}$$

$$\ddot{\boldsymbol{\eta}} = -\delta\dot{\boldsymbol{\omega}} - (K\boldsymbol{\eta} + C\dot{\boldsymbol{\eta}}) \tag{2.15}$$

Here, J_{mb} represents the inertia of the main body, which is related to the inertia tensor of the full body J using the rigid-flexible coupling matrix δ by $J = J_{mb} + \delta^T\delta$. The matrices K and C are the stiffness and damping matrix, respectively. They depend on the damping ratios and natural frequencies as $C = \text{diag}\{2\zeta_1\omega_1, \ldots, 2\zeta_N\omega_N\}$ and $K = \text{diag}\{\omega_1^2, \ldots, \omega_N^2\}$ where the pair (ζ_k, ω_k) are respectively the damping ratio and natural frequency of mode k.

Comparing equation (2.2) with equation (2.14), if δ is a zero matrix (no coupling between the rigid and flexible dynamics, which also implies $J = J_{mb}$), the flexible dynamics reduce down exactly to the rigid

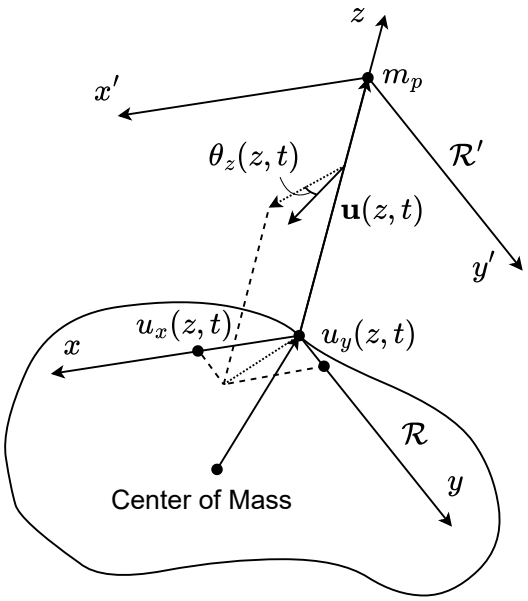


Figure 2.4: Spacecraft with a flexible boom, reproduced from Gennaro [117].

dynamics. Furthermore, K and C fully describe the vibrational behavior in terms of ζ_i and ω_i for all $i = 1, \ldots, N$. These can be derived from the vibrational eigenmodes [113], for example by using a finite element analysis [118]. The masses and inertia of the flexible components are accounted for by δ.

As the goal is to compare a rigid scenario and a flexible scenario, the parameters of the flexible model have been set to resemble the rigid model as closely as possible. The case that is under investigation can be described as a chaser that has grabbed the target using robotic arms. The chaser and target are considered to be rigid bodies, while the robotic arm connecting the two can flex to some degree. The dynamics should be representative for cases like these, but are not constrained to one specific case. This means that the matrix J is still set to Envisat's inertia tensor, the sampling frequency is still 1 Hz, and the maximum control torque is still 200 Nm, to make comparisons between the rigid and flexible case easier.

The inertia tensor of the spacecraft Gennaro uses for his analysis is roughly 225 times smaller than Envisat's inertia tensor. The matrix δ used by Gennaro is hence scaled up by a factor of $\sqrt{225} = 15$. This results in the following final matrix:

$$\delta = \begin{bmatrix} 96.84555 & 19.1721 & 32.34435 \\ -18.84285 & 13.7634 & -25.0896 \\ 16.75305 & 37.33515 & -12.5511 \\ 18.54555 & -39.8715 & -16.87545 \end{bmatrix} [\sqrt{\text{kg}}\,\text{m}] \tag{2.16}$$

The damping ratios were unchanged from Gennaro's analysis. The natural frequencies were scaled to be in the same order of magnitude as natural frequencies that could be expected for a flexible target-chaser combined body, using order-of-magnitude analysis. The robotic arm is on the order of magnitude of a meter in length. If the robotic arm is assumed to be an Euler-Bernoulli flexible beam, using a Young's modulus of 100 GPa (order of magnitude of the stiffness of carbon fibre, a material used on space manipulators such as the canadarm 2 [119]), and representative dimensions and inertias, the natural frequency of the beam in bending is on the order of 0.1 rad/s. The natural frequencies used by Gennaro are on the order of approximately 1 rad/s, so these are multiplied by 0.1 to get the natural frequencies used in this research. As a result, the damping ratios $\boldsymbol{\zeta}$, natural frequencies $\boldsymbol{\omega_n}$, and corresponding K and C matrices are:

$$\boldsymbol{\zeta} = \begin{bmatrix} 0.005607, .00862, 0.01283, 0.02516 \end{bmatrix}^T \tag{2.17}$$

$$\boldsymbol{\omega_n} = \begin{bmatrix} 0.07681, 0.11038, 0.18733, 0.25496 \end{bmatrix}^T [\text{rad/s}] \tag{2.18}$$

$$K = \begin{bmatrix} 0.00589978 & 0 & 0 & 0 \\ 0 & 0.01218374 & 0 & 0 \\ 0 & 0 & 0.03509253 & 0 \\ 0 & 0 & 0 & 0.0650046 \end{bmatrix} \tag{2.19}$$

$$C = \begin{bmatrix} 0.00086135 & 0 & 0 & 0 \\ 0 & 0.00190295 & 0 & 0 \\ 0 & 0 & 0.00480689 & 0 \\ 0 & 0 & 0 & 0.01282959 \end{bmatrix} \tag{2.20}$$

The decision to scale Gennaro's values using order-of-magnitude analysis instead of deriving exact values was driven by two considerations. The first is that as long as the dynamics are representative, but not necessarily representing an actual case, the research question can still be answered. Scaling Gennaro's values achieves this. The second is a time consideration, to spend the time available for the project on other topics that are more relevant to the research at hand (reinforcement learning in spacecraft control) instead of an in-depth (vibrational) analysis.

It should be noted that all other environment settings of the flexible spacecraft model are the same as the rigid model. This includes the numerical integration settings, with the timestep of 1/60 s. This timestep ensures the vibrations are accurately modelled without running into sampling issues. However, the agent is still modelled at 1 Hz. Given the natural frequencies, the vibrations are at least 24 times smaller than 1 Hz. The agents should hence be able to dampen all the vibrations. Checking whether they can actually do this is hence a point of interest.

2.3. Reinforcement learning setup

Simply inserting an off-the-shelf reinforcement learning algorithm in the place of the controller in figure 2.2 does not suffice for the purposes of this research. This is because of the different variables the studies will investigate, but also because of the nuances that make certain reinforcement learning agents performant or not. This section explains how the RL agents are implemented in a training loop, so that their performance can be assessed and compared.

Furthermore, the settings of the agents' learning process, which include the agents' hyperparameters and learning parameters such as maximum number of timesteps per episode, has a large effect on the performance of the agents. At the same time, the space of possible configurations is very large and highly multi-dimensional. As a result, this section also includes the settings around the learning process including an explanation and motivation. Then, the settings for the individual agents are discussed.

Settings other than the ones mentioned in this chapter are directly the ones used by the original author implementations. As an example, the reinforcement learning algorithms use feed-forward deep neural networks with two hidden layers, as specified in each of the original algorithm proposal papers [88][102][96][99].

2.3.1. Problem formulation

The first crucial definition in the formulation of the environment for the agent is that of the observation space. As discussed in chapter 1, reinforcement learning algorithms are generally built upon the assumption of a Markov Decision Process [120]. However, they can also perform well in a partially observable MDP.

The observation vector is taken as the feature vector for the reinforcement learning agent's neural networks. In the baseline scenario of this research, the observation vector is comprised of the quaternion attitude and the attitude rates, as measured by perfect sensors. The baseline case is formulated as a fully-observable MDP. The reward function, which will be described in more depth in the following subsection, includes a bonus in case the spacecraft's state has improved with respect to the previous timestep, based on the scalar part of the quaternion $q_{e,4}$. To ensure the system still has the Markov property, $q_{e,4}$ of the previous timestep is also appended to the feature vector, resulting in the baseline feature vector for the rigid baseline at timestep t_k as:

$$\boldsymbol{o}(t_k) = \boldsymbol{o}_k = \begin{bmatrix} q_{e,1_k} & q_{e,2_k} & q_{e,3_k} & q_{e,4_k} & \omega_{e,1_k} & \omega_{e,2_k} & \omega_{e,3_k} & q_{e,4_{k-1}} \end{bmatrix}^T \tag{2.21}$$

In the flexible case, the baseline experiment is the same as the rigid case, even though the system variables now include the modal coordinates and their derivatives. This means that the baseline flexible case is a partially observable one.

Three variations to the observation space are investigated. The first two are to replace the four quaternion components with either the three Tait-Bryan angles or the three modified rodrigues parameters. The third variation only exists in the flexible case, and is the fully observable case. In the third case, the observation vector given the parameters set in section 2.2.5 (with four modal coordinates) is set as:

$$\boldsymbol{o}(t_k) = \boldsymbol{o}_k = \begin{bmatrix} \boldsymbol{o}_{r_k} & \boldsymbol{o}_{f_k} & q_{e,4_{k-1}} \end{bmatrix}^T \tag{2.22}$$

$$\boldsymbol{o}_{r_k} = \begin{bmatrix} q_{e,1_k} & q_{e,2_k} & q_{e,3_k} & q_{e,4_k} & \omega_{e,1_k} & \omega_{e,2_k} & \omega_{e,3_k} \end{bmatrix} \tag{2.23}$$

$$\boldsymbol{o}_{f_k} = \begin{bmatrix} \eta_{e,1_k} & \eta_{e,2_k} & \eta_{e,3_k} & \eta_{e,4_k} & \dot{\eta}_{e,1_k} & \dot{\eta}_{e,2_k} & \dot{\eta}_{e,3_k} & \dot{\eta}_{e,4_k} \end{bmatrix} \tag{2.24}$$

It should be noted that the environment always simulated the kinematics using quaternions, even if Tait-Bryan angles or MRP representations were used in the observations. In these cases, the quaternions were simply converted to these other representations using the following coordinate transformations respectively before generating the observation vector [110]:

$$\begin{aligned} \phi &= \operatorname{atan2}\left(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2)\right) \\ \theta &= \arcsin\left(2(q_w q_y - q_z q_x)\right) \\ \psi &= \operatorname{atan2}\left(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2)\right) \end{aligned} \tag{2.25}$$

$$\mathbf{p} = \frac{\begin{bmatrix} q_x & q_y & q_z \end{bmatrix}^T}{1 + q_w} \tag{2.26}$$

The second major definition is that of the action space. While related work by Elkins has successfully used a discrete action in the past [92], the focus of this work is on continuous control. Initial experiments indicated that performance for an arbitrary symmetric action space $[-u_{\text{max}}, u_{\text{max}}]$ was sometimes very poor. Hence, the last operation within a policy network was set to a hyperbolic tangent activation layer, constraining the commanded action to between $[-1, 1]$. Outside of the policy network, the action would be scaled back to the original $[-200, 200]$ Nm action space in a linear fashion.

2.3.2. Reward function specification

The reward signal is what the reinforcement learning agent ultimately uses to learn, and tries to maximize. Since the reward function is usually set manually by humans, human biases are inherent to these signals. The solution to this problem is called reward function shaping, and is still an active area of research [121][122].

Optimizing the reward function for reinforcement learning performance is outside of the scope of this research. However, investigating the effects of often employed strategies in reward function shaping is within the scope, as these can guide future work on designing an optimal reward function.

As a baseline, the reward function used by the related work by Elkins is used [93]. He describes a reward function with four distinct notable properties:

1. A shaped reward function that is shaped exponentially towards the goal state of zero attitude error. Larger rewards are given if the state is closer to the goal state, incentivizing the agent to reach this region of the state-space as quickly as possible to maximize the total discounted reward (equation (1.1)).
2. A penalty for control effort, linearly proportional to the norm of the commanded control torque vector. This discourages the agent from always only choosing agents near the boundary of the action space. The intent is for the agent to learn a policy within the subset of optimal policies that minimizes the control effort required. This translates to lower propellant consumption or less frequent desaturation procedures, depending on the type of control actuator used.

3. A penalty if the current state is worse than the previous state (attitude error is further from zero). This further incentivizes the agent to continuously improve the state, reaching the target state as quickly as possible.
4. A large bonus in case the attitude error is within the requirement (0.25° in their case). This large bonus incentivizes the agent to stay within the requirements, which stabilizes the spacecraft around the target attitude.

They hence calculate the reward using:

$$r_{a,t} = \begin{cases} \exp\left(\frac{-\phi_t}{0.14 \cdot 2\pi}\right) - 0.5\frac{\|\boldsymbol{M}_t\|}{\|\boldsymbol{M}_{\max}\|} & \phi_t \leq \phi_{t-1} \\ \exp\left(\frac{-\phi_t}{0.14 \cdot 2\pi}\right) - 0.5\frac{\|\boldsymbol{M}_t\|}{\|\boldsymbol{M}_{\max}\|} - 1 & \text{otherwise} \end{cases} \tag{2.27}$$

$$r_t = \begin{cases} r_{a,t} + 9 & \text{if } \phi_t \leq 0.25^\circ \\ r_{a,t} & \text{otherwise} \end{cases} \tag{2.28}$$

ϕ_t represents the rotation around the euler axis that together represent the rotation error from the target, meaning that the goal state for ϕ should be 0. It is calculated by the inverse of the definition of the scalar part of the quaternion (see equation (2.5)), as: $\phi = 2\arccos(q_{e,4})$. In equation (2.27), the factor $0.14 \cdot 2\pi$ in the denominator ensures the exponential part of the reward is within approximately $[0, 1]$, and determines the slope of the reward as a function of $\phi \in [0, 2\pi]$. The fraction $\frac{\|\boldsymbol{M}_t\|}{\|\boldsymbol{M}_{\max}\|}$ is within $[0, 1]$, and is scaled by a factor -0.5. The penalty in case the state is worse than the previous state ($\phi_t > \phi_{t-1}$) is set to -1. Finally, in equation (2.28), the bonus for a state that is within the pointing requirement of 0.25° (the requirement used by Elkins) is $+9$.

As explained, this reward function is also used as a baseline in this research. However, for this case the baseline requirement is 1° instead of 0.25°, as explained in section 1.3.2. Furthermore, of interest is how these different components to the reward function influence the learning process. Hence, the generalized reward function used in this research is formulated as:

$$r_{a,t} = \begin{cases} \exp\left(\frac{-\phi_t}{0.14 \cdot 2\pi}\right) - c_a\frac{\|\boldsymbol{M}_t\|}{\|\boldsymbol{M}_{\max}\|} & \phi_t \leq \phi_{t-1} \\ \exp\left(\frac{-\phi_t}{0.14 \cdot 2\pi}\right) - c_a\frac{\|\boldsymbol{M}_t\|}{\|\boldsymbol{M}_{\max}\|} - c_p & \text{otherwise} \end{cases} \tag{2.29}$$

$$r_t = \begin{cases} r_{a,t} + 9 & \text{if } \phi_t \leq {c_r}^\circ \\ r_{a,t} & \text{otherwise} \end{cases} \tag{2.30}$$

where coefficients c_a, c_p, and c_r represent the action penalty multiplier, the penalty if the state does not improve, and the requirement in degrees, respectively, and have the default values 0.5, 1, and 1.

It should be noted that Elkins set up their environment to give a different reward upon episode termination. An episodes terminates due to a too large rotational rate, after which a large negative reward of -25 is given. This was also implemented in the reward function used in this research, with the termination criterion of 0.5π rad/s. This was done to discourage the agent to explore areas of the state-space with an exceedingly large rotational rate, which is counterproductive to convergence.

Elkins also implemented different rewards upon episode truncation. After 500 agent timesteps (in this research sampled in time intervals of 1 second/at 1 Hz), the episode truncates. In case the agent is within the required angular error, the reward in Elkins's environment is set to +50, and is 0 otherwise. They motivate this by stating that this further incentivizes the agent to stabilize within the requirements.

This truncation reward was not used within this research, as it takes away the Markov property, unless for example the time step is given in the observation vector. This decision was made because the agent has to ultimately control the spacecraft attitude in a continuous manner instead of an episodic manner. It is further motivated by the hypothesis that allowing the agent to do proper bootstrapping in the episodic case will result in better continuous performance. The motivation for this is illustrated by an example.

In this example, it is assumed that upon truncation, the agent is within the requirement and would stay there in case the episode were not truncated. In such a case, the lower bound on the instantaneous

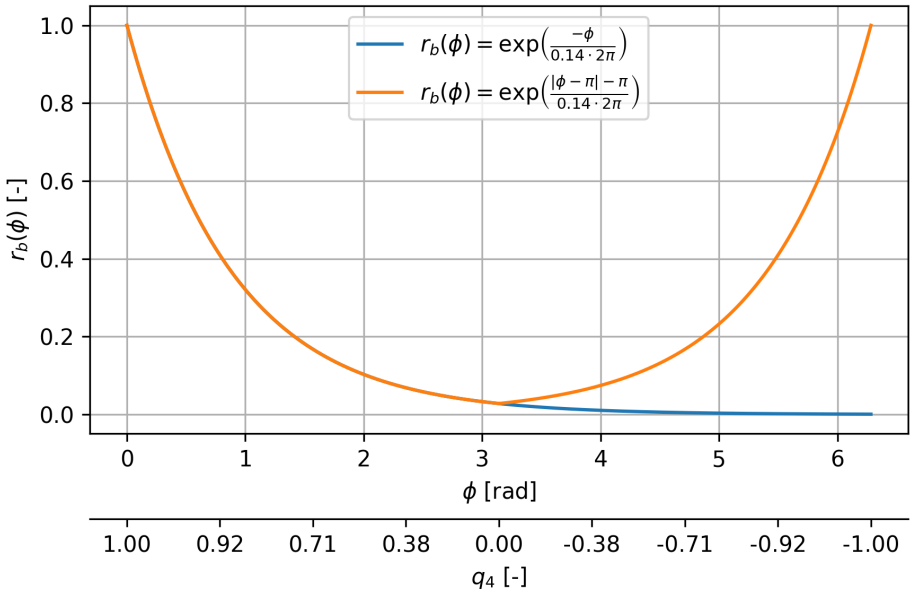


Figure 2.5: Exponential part of the reward function, original (asymmetric, blue) and alternative (symmetrical around $\phi = \pi$, orange) variants.

reward is 7.5, based on equation (2.30) for the default values of c_a and c_p. If the environment is assumed to be continuous instead of episodic and bootstrapping is used, the lower bound on the expected reward to go (as the right-hand term in the Bellman equation, see equation (1.5)) can be calculated as:

$$G = \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} = \gamma \cdot \frac{7.5}{1-\gamma} \tag{2.31}$$

which for $\gamma = 0.99$ works out to $G = 742.5$. A positive reward of +50 at the instant of truncation would hence only increase the bootstrapped value of G by 6.7%, while preventing the critic from ever converging to a loss of 0 due to the loss of the Markov property and hence increasing variance.

One note about the reward function described in equation (2.30) is that it rewards $\phi \to 0$ (equivalent to $q_4 \to 1$) more, while not rewarding $\phi \to 2\pi$ (equivalent to $q_4 \to -1$), even though both would push the environment towards zero attitude error. This touches upon a deeper property of quaternions, which is the 'double cover' property. The double cover property means that a versor $\boldsymbol{q}$ and $-\boldsymbol{q}$ represent the exact same rotation matrix. Intuitively, this can be interpreted by the fact that a rotation around Euler axis $\boldsymbol{e}$ by angle ϕ is equivalent to rotating about the opposing $-\boldsymbol{e}$ axis by rotation angle $-\phi$. This is also clear from equation (2.5) as $\boldsymbol{q}(\boldsymbol{e}, \phi) = \boldsymbol{q}(-\boldsymbol{e}, -\phi)$.

A variant on the reward function was hence considered. In this variant, the exponential term of the reward r_b in equation (2.29), where $-\phi$ in the exponent was replaced by $|\phi - \pi| - \pi$. This results in the figure shown in figure 2.5.

It was found that this alternate reward function drastically reduced learning stability, resulting in a higher probability of policy collapse during training and also generally resulted in lower pointing accuracy. However, precisely because of the double cover property, the representation of a rotation by either $\boldsymbol{q}$ or $-\boldsymbol{q}$ is an arbitrary decision. The initial state is hence always represented in what is called the canonical form, in which $q_4 \geq 0$. This minimizes the distance $q_4(t_0) - q_{4,\text{goal}}$, in case $q_{4,\text{goal}} = 1$. Hence, the original asymmetric reward function was used instead of the symmetric-around-π alternative reward function.

Apart from the reward function inspired by Elkins, a reward function similar to a more conventional control cost function was also investigated. A quadratic cost function, similar to one used for LQR control, was tried:

$$r = \boldsymbol{o}^T Q \boldsymbol{o} + \boldsymbol{u}^T R \boldsymbol{u} \tag{2.32}$$

with $\boldsymbol{o}$ as the observation and $\boldsymbol{u}$ as the commanded control torque. A problem with this formulation is that as the orientation error goes to zero, the reward hardly changes. This can be seen if the derivative $\frac{\partial r}{\partial \phi}$ is considered. When combining equation (2.32) with equation (2.21) and equation (2.5), it can be seen that this derivative goes to zero as well (due to the cosine term in equation (2.5)). This means that a large improvement in error angle $\partial\phi$ (which is sought) only results in a minimal improvement in the reward ∂r, which limits the incentive for the agents to further improve pointing performance.

Indeed, using this quadratic reward function in early test experiments resulted in the agents being able to stabilize the attitude close to the target orientation, but with a non-zero (and outside of the requirements) steady-state error. While there are ways to modify the quadratic reward function to reduce this behavior, such modifications will look a lot like the exponential term in equation (2.29). The variable coefficients in this reward function already allow for the comparison of the effects that different components of the reward function will have on the learning process of the agents. Literature, such as the work by Engel and Babuška [123], also suggests that a quadratic reward function, with parallels to LQR control, are not the best type of reward function for reinforcement learning. While they conclude this for different reasons, both these considerations were cause for not further considering a quadratic reward function in the final experiments.

2.3.3. Learning setup

To conduct a valid comparison between different agents, the environment and learning variables are kept equal as much as possible. The first of these is the interaction with the environment. At the core, the environment is simulated in an episodic fashion.

Every episode starts with a random initial state. The initial quaternion is set by sampling a random rotation matrix $C \in SO(2)$, and converting the matrix to a unit quaternion by using:

$$\begin{bmatrix} 1+2C_{11}-\operatorname{tr}C \\ C_{12}+C_{21} \\ C_{13}+C_{31} \\ C_{23}-C_{32} \end{bmatrix} = 4q_1\boldsymbol{q}, \quad \begin{bmatrix} C_{21}+C_{12} \\ 1+2C_{22}-\operatorname{tr}C \\ C_{23}+C_{32} \\ C_{31}-C_{13} \end{bmatrix} = 4q_2\boldsymbol{q}$$

$$\begin{bmatrix} C_{31}+C_{13} \\ C_{32}+C_{23} \\ 1+2C_{33}-\operatorname{tr}C \\ C_{12}-C_{21} \end{bmatrix} = 4q_3\boldsymbol{q}, \quad \begin{bmatrix} C_{23}-C_{32} \\ C_{31}-C_{13} \\ C_{12}-C_{21} \\ 1+\operatorname{tr}C \end{bmatrix} = 4q_4\boldsymbol{q} \tag{2.33}$$

Which equation is used depends on whether $\operatorname{tr}C$ is larger than each C_{ii} for $i = 1, 2, 3$. If it is larger the equation for q_4 is used, else the equation for q_i is used depending on which C_{ii} is largest. This selection of equation is done to limit numerical errors [110]. As discussed previously, the resulting quaternion was set to the canonical form, resulting in $q_4(t_0) \geq 0$. Furthermore, as also discussed earlier, the initial attitudes rates were set to zero. In case of the flexible model, the generalized modal coordinates and their derivatives were also initialized with zeros.

The second large consideration for episodic learning is the amount of steps per episode. This was set at 500 agent samples per episode. The agent sampling rate was set to 1 Hz, per section 2.2.2, resulting in a maximum episode length of 500 seconds. 500 steps was chosen because given the system time scale (driven by the control torque and inertia, as also explained in section 2.2.2) allows a tuned controller to reach a steady-state around the target orientation after between 50 and 200 seconds. Hence, given 500 time steps, an RL agent has sufficient time steps to explore a large section of the state space and to potentially converge to the target orientation. At the same time, a partially trained agent is prevented by the episode truncation from spending a disproportionate amount of time in the section of the state-space that is near the target orientation, which ensures the agent learns from a wider distribution of samples (supporting more exploration), reducing the probability that the agent gets stuck in a local optimum.

The third and fourth fundamental learning process considerations are the total amount of training iterations and the amount of runs per experiment. These have been set depending on which algorithm is used for the agent, so will be discussed in the next section.

The method for optimizing hyperparameters is the final important learning setting. The hyperparameters themselves, including default values, are discussed in the next subsection. However, the way these are optimized is generalized. The hyperparameters are sampled using a tree-structured Parzen estimator. This was done because it is the default optimization algorithm that is used by the chosen optimization software, Optuna [124], and they note that this algorithm tends to give better results than other algorithms (also random sampling) in case compute power is limited. Since achieving the most efficient optimization possible is not the focus of this research, this recommendation has simply been adopted.

2.3.4. Agent types and hyperparameters

As discussed in chapter 1, four different agents will be investigated: PPO, SAC, TD3, and TD7. For the former three, the implementation by the software package Stable-Baselines3 [125] is used. This was done because this implementation is a versatile implementation that eased the implementations of other aspects of this research, and because of the extent to which it is used by the others, which contributes to the reproducibility of the results in this research. The TD7 algorithm has been implemented on top of the Stable-Baselines3 programming interfaces, based on the original author's implementation [99].

Some hyperparameters are shared between these algorithms, but others are unique to the respective algorithm. First, the general hyperparameters are discussed.

All of these algorithms are actor-critic algorithms. This means that they have two (in case of TD7 three, this will be discussed later) feedforward neural nets. These nets are defined by the amount of nodes in a hidden layer, the amount of hidden layers, and the activation function between these layers. All of networks are iterated using a batch of environment interaction samples of a certain batch size at a time, under influence of the optimization algorithm. The step size of the optimizer is influenced by a learning rate. Finally, the learning process is influenced by the selection of the discount factor.

The baseline hyperparameters are listed in table 2.2. The hyperparameters, when possible, are taken from Elkins et al [93], so that insights between this research and theirs can be compared. Other hyperparameters are taken from the Stable-Baselines3 implementation of the algorithms, which have been optimized for a wide range of environments [126], or in case of TD7, taken from the original paper [99].

Stable-Baselines3, and hence all the agents used in this research, have been implemented using PyTorch [127]. Weights of the neural networks are initialized using the default PyTorch implementation, which is based on Kaiming uniform initialization [128].

This random initialization of weights influences the learning process and the final performance of the trained agent. To be able to draw conclusions about specific settings or architectural considerations in the learning setup, the effects of this randomization need to be minimized. Hence, every experiment is run multiple times. The computational power available for this project is limited, which is constraining the total amount of runs. For every study, roughly the same amount of runs was selected. Some of the studies assess the effects of changing a single parameter by a fixed amount, while others use Optuna to sample hyperparameter values across multiple experiments, resulting in different results every experiment. In the former case, an experiment was repeated at least 7 times, while an experiment with a single sampled hyperparameter was run 2 times (but at least 10 different samples were tried).

The total training steps were, as discussed, differed per agent. This was done to keep the total computational time per run limited, while still obtaining valid results. The cut-off was set at a point after which the performance of the baseline agent did not change meaningfully anymore, based on multiple early experiments. This is further illustrated in figure 2.6.

PPO on average required the most environment interactions by a large margin (but was also the fastest by far). Hence, the total steps done for this algorithm was set at 3.5M steps. The computational time required for SAC and TD3, as well as the point after which no meaningful change occured anymore, was quite similar. Both have been set to stop training after 400,000 steps. TD7 required significantly more computational resources, but also often was more sample efficient. It's maximum training steps was set to 250,000.

Based on initial tests, the sample efficiency of some of the algorithms was determined to be (much) larger than others, warranting evaluation steps at a higher frequency. PPO was evaluated every 10000

Table 2.2: Baseline hyperparameters for the reinforcement learning agents

Category	Hyperparameter	Value
Shared	Discount factor γ	0.99
	Learning rate	0.0003
	Layer count	2
	Optimization algorithm	Adam
PPO	Batch size	64
	Steps per training iteration	2048
	Amount of training epochs	10
	GAE-λ	0.95
	Clip range	0.2
	Value function coefficient	0.5
	Maximum gradient norm	0.5
	Target KL divergence	Unused
	Activation function	ReLU
	Node counts in critic hidden layers	400-300
	Node counts in actor hidden layers	400-300
	Exploration noise σ	0.0
SAC	Batch size	256
	Polyak update coefficient τ	0.005
	Train frequency	1 (step)
	Gradient steps per train iteration	1
	Target update interval	1
	Activation function	ReLU
	Node count in all hidden layers	256
	Amount of critics	2
	Exploration noise σ	0.0
TD3	Exploration noise σ	0.1
	Batch size	256
	Polyak update coefficient τ	0.005
	Train frequency	1 (step)
	Gradient steps per train iteration	1
	Policy delay	2
	Target policy noise	0.2
	Target noise clip	0.5
	Activation function	ReLU
	Node counts in critic hidden layers	400-300
	Node counts in actor hidden layers	400-300
	Amount of critics	2
TD7	Exploration noise σ	0.1
	Batch size	256
	Checkpointing	Enabled
	Late assessment episodes	20
	Criteria reset weight	0.9
	Target update frequency	250
	Probability smoothing α	0.4
	Policy delay	2
	Target policy noise	0.2
	Target noise clip	0.5
	Activation function	ELU
	Node count in all hidden layers	256
	Amount of critics	2
	Encoder hidden layers	2

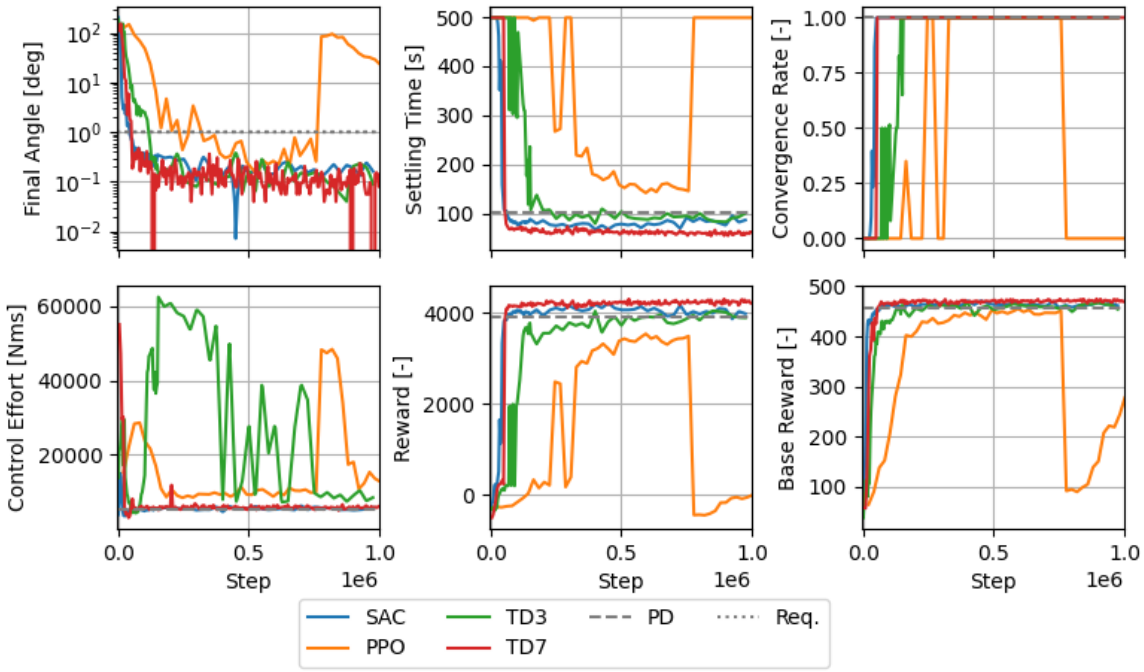


Figure 2.6: Demonstration of the diminishing returns for longer training times for the four algorithms. The algorithms no longer show continued improvements in performance after a relatively early timestep. Results are shown for a single training process of a baseline agent in the rigid environment, without smoothing applied.

steps, and the off-policy algorithms every 2500 steps.

2.4. Learning process assessment

During the learning process, the agents are assessed at regular intervals. This allows for the characterisation and reliability of the learning process, which is important for understanding and explaining the final results, which are explained in the next section.

Tracking learning progress in machine learning is mainly done [99] by visualising the process in a learning curve, showing total episode reward over time. Often, every point in this curve is averaged over a set of episodes. Furthermore, the curves themselves are the average of multiple runs of the same experiment. Finally, the curves are smoothed.

An important note for this research compared to some other research is that multiple reward functions are used, making the comparison of total episode reward over time invalid. Hence, during the training of an agent, a base reward is also calculated and saved. This is purely based on the actual error angle, and is calculated using:

$$r_b = \exp\left(\frac{-\phi_t}{0.14 \cdot 2\pi}\right) \tag{2.34}$$

Evaluations of agents during training done by running the agent for 20 episodes. During these episodes, the following statistics are saved:

1. Total undiscounted episode reward
2. Total undiscounted episode base reward
3. Total episode control effort
4. Episode length
5. Settling time

6. Angular error at the final episode timestep $\phi(t_{\text{end}})$
7. Smallest angular error of the entire episode ($\min \phi(t_i)$ for $i = 0, \ldots, T$)
8. Convergence (boolean)

Of these metrics, except for the convergence flag, the means, standard deviations, minima, and maxima are saved. Convergence is defined as the case in which the angular error and rate on the final timestep of the episode are below the requirements (1° and 0.1°/s respectively, as discussed in section 1.3.2). The settling time is set based on the convergence: if the episode converged, the settling time is the last moment after which the angular error and rates stay within the requirements. If it is not converged, the settling time is set to the episode length. Hence, the average settling time for an agent that does not converge is 499 s, the maximum episode length.

The total episode control effort (in Nms) is calculated by taking the (numerically calculated) integral of the norm of the commanded control torque vector over the entire episode:

$$u_{\text{tot}} = \int_0^{t_{\text{end}}} \|\boldsymbol{u}(t)\| dt \tag{2.35}$$

All the mentioned performance metrics are compared to the performance metrics of a PD controller. This is done to provide a well-understood frame of reference of the performance of the trained agents, while being easily replicable. These PD controllers were tuned by hand, for the same environment setup as the reinforcement learning agents. This means that the output of the PD controller was clipped to the control torque limits. Furthermore, since the environments are defined to receive a scaled commanded control torque in $[-1, 1]$, the clipped output was linearly rescaled to this range.

The tuning of the PD controllers was done for both the rigid and flexible cases separately. The controller, which was implemented as a full PID controller, is defined as:

$$u(t) = K_p \boldsymbol{o}_e(t) + K_i \int_0^t \boldsymbol{o}_e(\tau) d\tau + K_d \dot{\boldsymbol{o}}_e(t) \tag{2.36}$$

with gain matrices K_p, K_i, and K_d. Furthermore, $\boldsymbol{o}$ is the observation vector, as defined and described in section 2.3.1. For the rest of this research (except in the verification section 3.2), the integral gain K_i is the zero matrix. The proportional and derivative gain matrices used were set to:

$$K_p = \begin{bmatrix} k_{p,q} & 0 & 0 & 0 & k_{p,\omega} & 0 & 0 \\ 0 & k_{p,q} & 0 & 0 & 0 & k_{p,\omega} & 0 \\ 0 & 0 & k_{p,q} & 0 & 0 & 0 & k_{p,\omega} \end{bmatrix}, \quad K_d = \begin{bmatrix} k_{d,q} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{d,q} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{d,q} & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.37}$$

with the variable gains listed for both the rigid and flexible environments in table 2.3. $k_{d,q}$ was included (even though $k_{p,\omega}$ was also included) as in the tuning process the controllers with a non-zero $k_{d,q}$ performed better. This is likely a result of the coupling in equation (2.6).

It should be noted that the observation vector $\boldsymbol{o}$ in equation (2.36) was truncated to exclude $q_{e,4_{k-1}}$ and any modal coordinates, if applicable, leaving only the error quaternion and rate error in the observation vector.

Table 2.3: PD gains tuned for the rigid and flexible environments.

Gain	Rigid tuned	Flexible tuned
$k_{p,q}$	-1200	-625
$k_{p,\omega}$	-14400	-11440
$k_{d,q}$	-600	-440

The performance of these PD controllers, both for the intended environment and also the other environment variant, has been tabulated in table 2.4 for reference.

In table 2.4, it should be noted that when looking at the flexible environment but also in general, the base reward is not an adequate metric for performance. It is clear that even though the convergence

Table 2.4: PD controller performance in the different environments. The values after $\pm$ represent the standard deviation of the value over 200 randomly initialized episodes.

Environment	Rigid		Flexible	
Controller (PD)	Rigid tuned	Flexible tuned	Rigid tuned	Flexible tuned
Episode length	499±0	499±0	499±0	499±0
Episode reward	3942.2±114	3570.7±180	1657.3±615	3362.8±276
Episode base reward	458.3±10.2	440.8±12.9	448.4±10.7	442.1±13.3
Final angle [deg]	0±0	0±0	1.003±0.530	0.267±0.158
Best angle [deg]	0±0	0±0	0.036±0.029	0.066±0.036
Control effort [Nms]	4917.2±4272.7	3987.5±3140.7	38444.9±40223.2	4846.8±2582.4
Settling time [s]	97.5±12.9	150.5±19.0	499±0	175.1±61.3
Convergence rate	100%	100%	0%	100%

rate is 0%, the base reward is still relatively high at 448.4. However, when the PD controller is properly tuned for the flexible environment, the base reward drops down to 438.8, while practically all other performance metrics move in the direction that indicates better performance. Hence, it should only be used to compare between agents that have different reward functions but are otherwise identical, and only together with the other performance metrics.

The learning curves that will be shown are in some cases smoothed by using a 10-step window average. This is indicated if it is not the case.

The final important metric that is looked at is the computational time. This measurement simply measures the wall time elapsed between different milestones. This can either be the total trainign time, time per 100k steps, of the time at which the agent exceeds the mean total episode reward of the appropriate PD controller. This provides indication of the learning speed, sample efficiency, and computational cost. Note that the measurement of the computational time also includes the time required to simulate the spacecraft and to evaluate the agent. However, the spacecraft is simulated with the exact same numerical settings for each algorithm.

2.5. Robustness assessment

The appropriately tuned controllers of table 2.4 for the respective models already have a fairly good nominal performance, achieving fairly rapid settling times for a reasonable total control effort, with a 100% convergence rate. However, the research question investigates performance under the influence of inherent dynamic and kinematic uncertainties. In this section, it is explained how these uncertainties are modelled, and also implemented in the full model.

2.5.1. Uncertainty modelling using perturbations

In this research, the set of off-baseline experiments are called perturbed experiments, and the changes are called perturbations. These perturbations are only applied orthogonally, meaning that no two perturbations are applied at the same time. This was decided due to computational limitations.

The perturbations were implemented identically for both the rigid and the flexible environments. The perturbations that were investigated in this research are listed below.

1. Inertia scaling. The perturbation applied represents the sensitivity of a controller to different mass distributions. This was done by rescaling the inertia tensor. First, the inertia tensor was diagonalized:

$$J = PJ_dP^T \tag{2.38}$$

Here, J_d represents the inertia tensor in the frame of the body's principal axes, and P represents the rotation matrix between the body frame and the principal axes frame. This diagonalization is always possible, as an inertia tensor is positive-definite. Then, each component of the diagonal matrix J_d is scaled by a random scaling factor:

$$\tilde{J}_d = \text{diag}\{\alpha_1 j_1, \alpha_2 j_2, \alpha_3 j_3\} \tag{2.39}$$

where $\alpha_i \sim \mathcal{N}(1, \sigma_J^2)$ and j_i is the entry on the diagonal of J_d, for each $i = 1, 2, 3$. After rescaling this diagonal, the inertia tensor in the body frame is obtained again by using the inverse of equation (2.38), $\tilde{J} = P\tilde{J}_d P^T$.

The perturbation is hence characterized by σ_J. In the unperturbed case, it is set to 0. In the perturbed case, a value of $\sigma_J = 0.006$ is used, as the results from inertia estimation studies in an ADR context, such as Zhang et al [129] show that errors in inertia tensor estimation of around 0.6% are reasonable for this case.

2. Inertia rotation. This is the second perturbation that represents errors in the mass distribution estimation. For this perturbation, a random unit vector $\boldsymbol{e} \in S^2$ is sampled. Then, a random rotation angle ϕ is sampled from $\mathcal{N}(0, \sigma_R^2)$. This rotation axis and angle is then converted to the representative rotation matrix R. Finally, the inertia tensor is rotated by this rotation matrix:

$$\tilde{J} = RJR^T \tag{2.40}$$

In the unperturbed case, $\sigma_R = 0$, resulting in $R = \boldsymbol{I}_{3\times 3}$. In the perturbed case, a value of 0.19 rad is used, again based on Zhang et al [129].

3. Gyroscope noise. This perturbation simulates the presence of noise in gyroscope measurements, which is the assumed way of measuring the attitude rates for the spacecraft. The noise is modeled as Gaussian white noise, added to the true angular velocity readings:

$$\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega} + \boldsymbol{\varepsilon} \tag{2.41}$$

where $\varepsilon_i \sim \mathcal{N}(0, \sigma_\omega^2),\ i = 1, 2, 3$ represents the white noise vector, sampled every step. In the unperturbed case, $\sigma_\omega = 0$. For the perturbed case, $\sigma_\eta = 0.057$ rad/s is used to reflect realistic noise levels found in gyroscopic sensors, based on Thienel and Sanner [130].

4. Gyroscope bias - constant. This perturbation accounts for a constant bias present in the gyroscope measurements, which can result from sensor imperfections or calibration errors. The biased measurement $\tilde{\boldsymbol{\omega}}$ is modeled as:

$$\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega} + \boldsymbol{b} \tag{2.42}$$

where $\boldsymbol{b}$ is a constant bias vector. The value of $\boldsymbol{b}$ is chosen based on typical bias levels observed in real gyroscopes. Other studies use values around the order of magnitude of 1 deg/s [130][131], but it can also be chosen semi-arbirarily [132]. In this research, $\boldsymbol{b} = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}^T$ which is sampled by $b_i \sim \mathcal{N}(0, \sigma_b^2),\ i = 1, 2, 3$. In the unperturbed case, $\sigma_b = 0°$, which results in $\boldsymbol{b} = \boldsymbol{0}$. In the perturbed case, $\sigma_b = 0.1°$. The bias is kept constant for the entire episode.

5. Gyroscope bias - random walk. This perturbation is introduced using equation (2.42) as well. However, in this case, the bias is time-varying, modeled as a random walk process. The bias is modelled using:

$$d\boldsymbol{b}_t = \sigma_b dW_t \tag{2.43}$$

where W is a Wiener process, characterized by $W_t - W_{t-1} \sim \mathcal{N}(0, \Delta t)$. Since the noise is only relevant in the sensors as seen in figure 2.2, the bias is propagated at the same frequency as the agent is sampled, so $\Delta t = 1$ s. σ_b characterizes the variance of the bias term. $\boldsymbol{b}(0) = 0$ is used as the initial condition. In the unperturbed case, σ_b = 0, meaning that $\boldsymbol{b}(t) = 0\,\forall\, t$. In the perturbed case, $\sigma_b = 0.057$, following Thienel and Sanner [130].

6. Torque misalignment. This perturbation represents misalignments in the actuation system, where the applied torques are not perfectly aligned with the intended control axes. The actual torque $\tilde{\boldsymbol{u}}$ applied modelled as:

$$\tilde{\boldsymbol{u}} = (R_u \boldsymbol{u}) \circ \tau + \boldsymbol{\eta} \tag{2.44}$$

Here, R_u is a rotation matrix representing the misalignment. $\circ$ represents the operator for element-wise multiplication of the vectors, τ represents the action scaling vector discussed in the next perturbation, and $\boldsymbol{\eta}$ represents the action noise, discussed in the perturbation after that.

R_u is sampled using the same process as perturbation number 2, inertia rotation, but using a rotation angle sampled from $\mathcal{N}(0, \sigma_u^2)$ instead. In the unperturbed case, R_u is the identity matrix,

as $\sigma_u = 0$. In the perturbed case, a value of $\sigma_u = 10^\circ$ is used. This value is quite large, but is set to investigate the effects of misalignments that cause a change in thrust on the order of a few percent, which is roughly the order of magnitude Visser et al found to be reasonable for the magnetorquers on GOCE [133]. Even through magentorquers are not representative at all of a maximum torque of 200 Nm, as discussed previously, the goal is to research a suitable scope of perturbations that give insights into the applicability of the RL agents for a wide range of possible missions and control architectures.

7. Action scale. This perturbation scales the control inputs, simulating variations in actuator effectiveness or changes in control gains. It is modelled by the τ term in equation (2.44). $\tau = \begin{bmatrix}\tau_1 & \tau_2 & \tau_3\end{bmatrix}^T$ is sampled by $\tau_i \sim \mathcal{N}(1, \sigma_\tau^2),\ i = 1, 2, 3$. In the unperturbed case, $\sigma_\tau = 0$. In the perturbed case, $\sigma_\tau = 0.03$, causing a change on the order of a few percent.
8. Action noise. This perturbation adds noise to the control inputs, reflecting imperfections in the actuation system. This is modelled by the term $\boldsymbol{\eta}$ in equation (2.44). The noise term is sampled every step as $\boldsymbol{\eta} \sim \mathcal{N}(0, \sigma_\eta^2 \boldsymbol{I})$. In the unperturbed case, $\sigma_\eta = 0$. In the perturbed case, $\sigma_\eta = 0.03\boldsymbol{u}_{\text{max}} = 6$ Nm, again causing a change on the order of a few percent.
9. Disturbance torque. This perturbation includes external torques acting on the spacecraft, such as those due to environmental effects like aerodynamic drag or gravitational torques. The disturbance torque $\boldsymbol{u}_d$ is added to the control torque, as previously shown in equation (2.3), repeated here:
$$\boldsymbol{M}_B = \tilde{\boldsymbol{u}} + \boldsymbol{u}_d \tag{2.3}$$
In the unperturbed case, $\boldsymbol{u}_d = \boldsymbol{0}$. The perturbed case is based on the work by Cao et al [134], where the disturbance torque is given by:
$$\boldsymbol{u}_d = \begin{bmatrix} -3 + 4\cos(0.2\pi t) - \cos(0.4\pi t) + 2\omega_1 \sin(0.11t) \\ 4 + 3\sin(0.2\pi t) - 2\cos(0.4\pi t) + \omega_2 \cos(0.11t) \\ -3 + 4\sin(0.2\pi t) - 3\sin(0.4\pi t) - 2\omega_3 \cos(0.11t) \end{bmatrix} d \tag{2.45}$$
Here, d is a scaling factor that Cao et al set to 10^{-4} Nm, while their maximum control torque was set to 0.5 Nm. In this case, the inertia tensor and maximum control torque are of a much larger order of magnitude. Hence, the value for d has been set to 0.04 Nm, to keep the ratio between maximum disturbance magnitude and maximum control torque magnitude the same as in Cao's work.
10. (Flexible only) full observability. This case was already briefly discussed in section 2.3.1. The full observability perturbation is the case in which the agent has access to the modal coordinates. Conversely, in the unperturbed, the modal coordinates are hidden to the agent, making the unperturbed flexible case a partially observable MDP.

The perturbation terms that model noise are randomly sampled every time step. Perturbations that are changes to the spacecraft model (such as the inertia perturbations) or control loop (bias in the sensors) are reset every new episode, so that each new episode the agent encounters a new (perturbed) spacecraft model.

All perturbations were either activate or not. No further activations, where for example the standard deviations mentioned above were set to a different non-zero value than the one mentioned, were tried. This is because the study does not focus on exactly quantifying the effect of one of these disturbances for the specific mission (environment and agent settings mentioned). This has two reasons. The first is a time constraint on the entire project, and the second is that the actual values for the perturbations highly depend on the specific mission, which is as stated before outside the scope of the project.

2.5.2. Domain randomization

A consideration that is fundamental to answering the research question is not only whether a controller trained on the baseline scenario is able to deal with the perturbations mentioned in the previous subsection. Rather, a key strength of reinforcement learning is its ability to generalize a good policy for a wide range of cases. To get an idea about the ability of the algorithms to generalize (and hence, becoming more robust), their training domain will be randomized, which should contribute to the transferability from simulation to the real world, per Tobin et al [135].

The domain randomizations that were implemented include all of the perturbations of the previous sections, meaning that these perturbations were applied during training time. These domain randomizations were also only applied orthogonally: one at a time per training process.

Of course, the idea is that the domain randomized agents will learn more generalized policies. It is suggested that these generalized policies are more robust than the baseline case. Hence, these domain randomized agents are also tested against the same robustness tests as the baseline agents.

2.5.3. Robustness sensitivity analysis

The research question aims to answer "to what extent" the RL agents are robust to the uncertainties, implemented using the perturbations of section 2.5.1. Providing an answer to this question is not straightforward however, which is why this research has opted to use a comparative method.

First, the robustness experiments (for all the perturbations, one by one, including the unperturbed case) are ran for the baseline agents. In this context, the baseline agent refers to one of the agents with default settings including hyperparameters, reward function, attitude representation, and observability.

Next, any variation in an agent is also evaluated using the robustness experiments (again, all the perturbations, including the unperturbed case). Between all the repetitions (to reduce randomness in the results, as mentioned in section 2.3.4) of the experiment the mean and standard deviation of every performance metric is taken.

Next, the mean and standard deviation of the varied agent performance metrics $\hat{x}_i$ and $\hat{\sigma}_i$ for each performance metric index i are rescaled to represent a percentage change of the mean baseline performance x_i:

$$\begin{aligned} \bar{x_i} &= \frac{\hat{x}_i - x}{x} \cdot 100\% \\ \bar{\sigma_i} &= \frac{\hat{\sigma}_i}{x} \cdot 100\% \end{aligned} \tag{2.46}$$

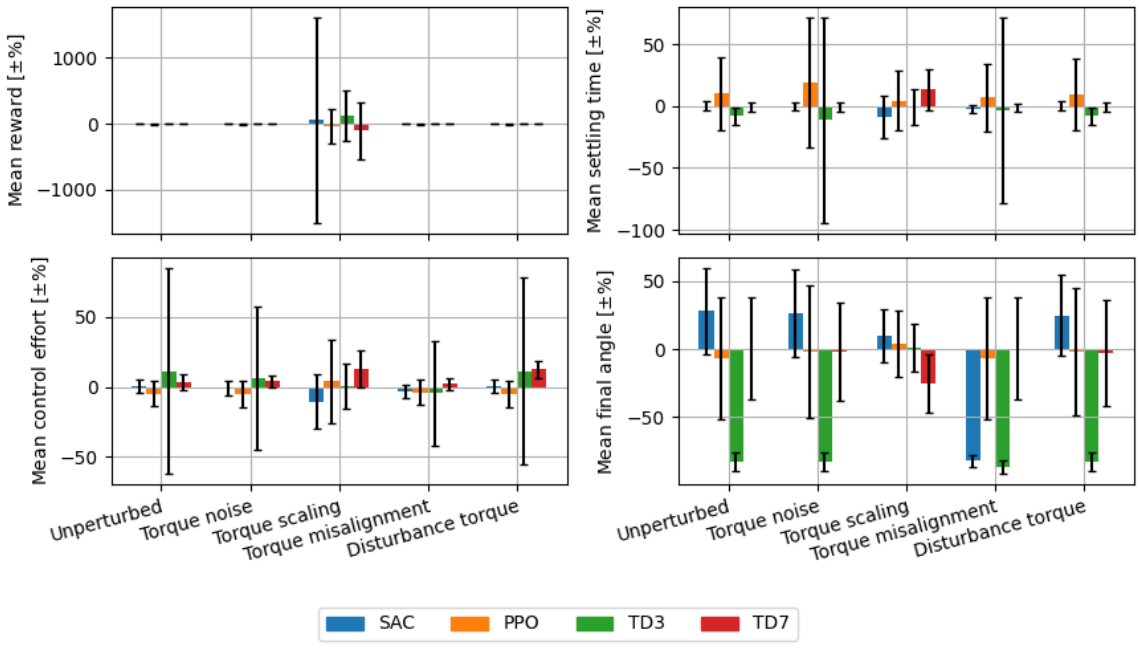


Figure 2.7: Example sensitivity analysis figure for the rigid environment, with inertia scaling as the variation to the agent. Each sub-figure shows a different performance metric, in this case (clockwise from top left) mean reward, mean settling time, mean final angle, and mean total control effort. The x-axis shows which perturbation is applied during evaluation, while the different bars represent the different algorithms that are used. The error bars indicate one standard deviation.

This percentage change $\bar{x_i}$ indicates how much better or worse the varied agent performs with respect

to the baseline agent. As a result, it estimates the sensitivity of the total agent performance to the variation. This differs per perturbation, and even per performance metric. Hence, these results will be shown in a bar chart like the one in figure 2.7. Note that the standard deviation error bars are applied in a symmetric way, but that this is not necessarily representative of the dataset. Also note that for the reward variations, it was deemed more useful to have different bars represent a different value for a coefficient (either c_r, c_a, or c_p, see section 2.3.2) all for the same RL algorithm, instead of showing multiple algorithms in a single figure.

3

Verification and Validation

Before the results of the research are presented, the implementation and validity of the methods are confirmed by the verification and validation efforts. Since multiple different reference works were adapted and combined as described in chapter 2, this resulted in a high degree of complexity. Due to this complexity, the focus of the main V&V efforts were also on the implementation of these combined methods.

In this chapter, first, the unit testing of important modelling components is discussed in section 3.1. Then, the PD controller is verified in section 3.2. Next, the custom implementation of the TD7 algorithm is verified in section 3.3. Next, both the rigid and the flexible spacecraft models are verified in section 3.4 and section 3.5 respectively. Finally, the full model is validated in section 3.6. Note that in reality, the rigid environment, including its verification and validation work, was completed before the TD7 algorithm was implemented and verified. Hence, in section 3.3, the environment should be considered verified, even though the environment verification work is shown afterwards.

3.1. Unit testing and assumptions

Several software packages are used off-the-shelf. Some functions have been implemented in a custom fashion. For both of these, it is important to reflect on their correctness, and not to assume without a solid rationale. In this section, first, the largest assumptions in this process are explained. Then, the coordinate transformation functions are discussed, followed by the verification of important implementation specific software elements.

3.1.1. Assumptions

All modelling efforts were completed in Python. This allowed for the usage of extensive and well-maintained libraries for a wide range of functionalities. The most important of these are NumPy [136], SciPy [137], Matplotlib [138], and PyTorch [127]. As these are widely regarded as the industry standard in Python or the machine learning community, these libraries are assumed to be correct.

Other libraries are also used. The libraries of note are Gymnasium [139], Optuna [124], and Stable-Baselines3 [125]. Gymnasium is used for the implementation of the environments, Optuna is used for the tuning of hyperparameters, and Stable-Baselines3 is used as the primary reinforcement learning library. These libraries depend on the libraries that were mentioned before. While these libraries are also widely used and actively maintained, these were screened by reviewing their source code. Otherwise, these were also assumed verified.

It should be noted that the implementations used for the algorithms PPO, SAC, and TD3 are part of Stable-Baselines3. These are hence also considered verified, but were also reviewed against the original author papers. The validation for these was done by using a CartPole environment (from Gymnasium). All environments converged quickly to a highly performant policy, as expected with this very simple environment.

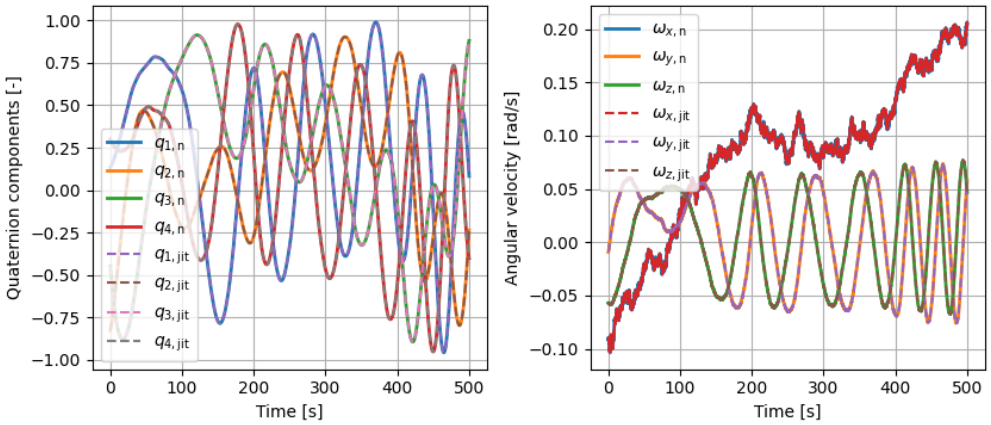


Figure 3.1: Rigid environment episode, JIT (subscript jit) and non-JIT (subscript n) version, for the same initial condition and random actions.

3.1.2. Coordinate transformations

A wide range of coordinate transformations was used. As mentioned in chapter 2, this includes transformations between quaternions and Euler angles and modified rodriguez parameters, and rotation matrices. These were implemented in a custom fashion, again based on the book by Markley [110] (and shown in equation (2.33), equation (2.25), and equation (2.26)). All conversions that were implemented were checked with success against the SciPy coordinate transformation functions. The custom implementation was done because it made dealing with the specific notation conventions used in this research easier.

3.1.3. Implementation specifics

The research project was limited in both schedule and computational time. Hence, efforts were done to speed up the models. The most important of these is the usage of Numba [140]. It allows for Just-In-Time (JIT) compilation of Python code, providing a notable speed increase after the compilation has completed. Unfortunately, more complex Python packages such as PyTorch are not supported or only supported partially. Hence, only the spacecraft models were JIT compiled. For the same initial condition and the same applied (random) torque, the JIT versions were compared against the non-JIT versions. The example for the rigid environment is visible in figure 3.1. To run 1M environment steps, including the initial compilation, the non-JIT version on the author's machine took 119 s, while the JIT version took 40 s, indicating that the JIT version is roughly 200% faster.

In chapter 2, it was mentioned that the environment numerical integration was done with a timestep of 1/60 s. Whether this makes the numerical error negligible, is an important assumption to verify. Hence, two versions of the environment were ran with the same initial conditions, and the same arbitrary constant torque of $\boldsymbol{u} = [1, -2, 1.5]$ Nm. The nominal environment ran with a timestep of 1/60 s, while the verification environment ran with 1/6000 s, a factor 100 more granular. Both were ran for 500 seconds, the result of which is shown in figure 3.2. When calculating the difference between the quaternions or rates, the difference for the quaternions was exactly 0 and for the rates exactly zero except for a couple of timesteps, where the error was on the order of magnitude of 10^{-12} rad/s. Because PyTorch is used, which on the author's hardware only supports 32-bit floating point numbers for the neural networks in the RL agents, the numerical error is negligible for a timestep of 1/60 s. This is because the 32-bit floating point number truncation error is much larger (the machine epsilon $\approx 1.12 \cdot 10^{-7}$).

3.2. PD controller verification

The reference PD controller also needs to be verified, as it is a custom implementation, for any comparison to be valid. It is implemented as a PID controller. Hence, verification is done by checking it against the PID controller implementation in Matlab using the Control System Toolbox add-on. Matlab and its Control System Toolbox are considered verified within this project.

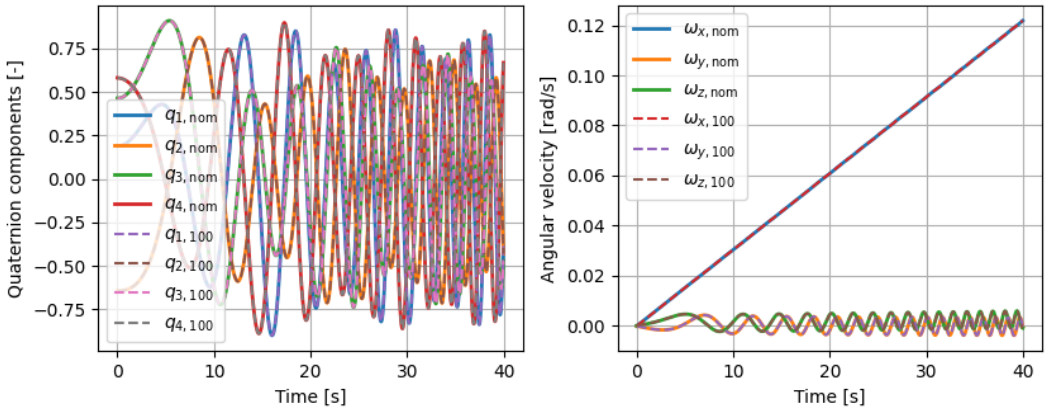


Figure 3.2: Verification of the numerical integration settings: nominal (subscript nom) environment with a timestep of 1/60 s and a 100x finer timestep environment (subscript 100) $dt = 1/6000$ s.

The verification was done for a simple spring-damper system, following a reference by the University of Michigan [141]:

$$m\ddot{x} + b\dot{x} + kx = F \tag{3.1}$$

where the variable x represents the position of the spring, and m, b, and k are the mass, damping, and stiffness constants respectively. These have been set to $m = 1$ kg, $b = 10$ Ns/m, and $k = 20$ N/m. This results in the transfer function:

$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20} \tag{3.2}$$

The implementation of this system in Python was done numerically, while in Matlab, the transfer function was used directly for the PID controller. The Python implementation was verified by checking the step response of the system, using the matlab code:

```
s = tf('s');
P = 1/(s^2 + 10*s + 20);
t = 0:0.01:2;
[y, tOut] = step(P, t);
```

The output `y` was compared to the output of the Python implementation, as shown in figure 3.3. After

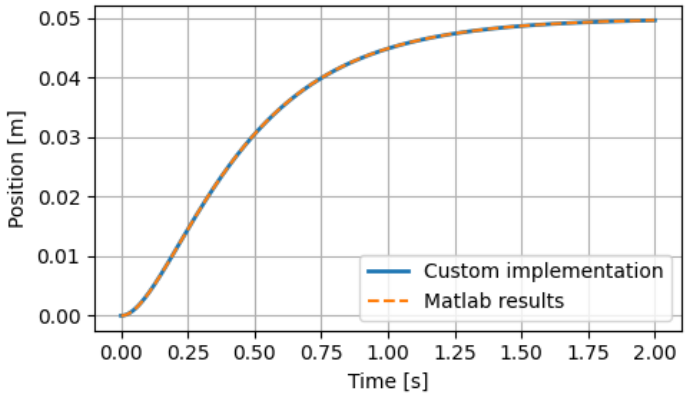


Figure 3.3: Simulation of a spring-damper system step response to verify the custom spring-damper system implementation.

the test system was verified, the PID was verified. This was done by setting the gain matrices in equation (2.36) to:

$$K_p = \begin{bmatrix} 350 & 0 \end{bmatrix}, \quad K_i = \begin{bmatrix} 300 & 0 \end{bmatrix}, \quad K_d = \begin{bmatrix} 50 & 0 \end{bmatrix} \tag{3.3}$$

as the observation vector of equation (2.36) for the numerical Python implementation of equation (3.1) is defined as $\boldsymbol{o} = \begin{bmatrix} x & \dot{x} \end{bmatrix}$. This was compared to the results of rest of the matlab script:

```
Kp = 350;
Ki = 300;
Kd = 50;

C = pid(Kp,Ki,Kd);
T = feedback(C*P,1);

[y_pid, tOut_pid] = step(T,t)
```

The results of this comparison are shown in figure 3.4. As the results match to within numerical errors,

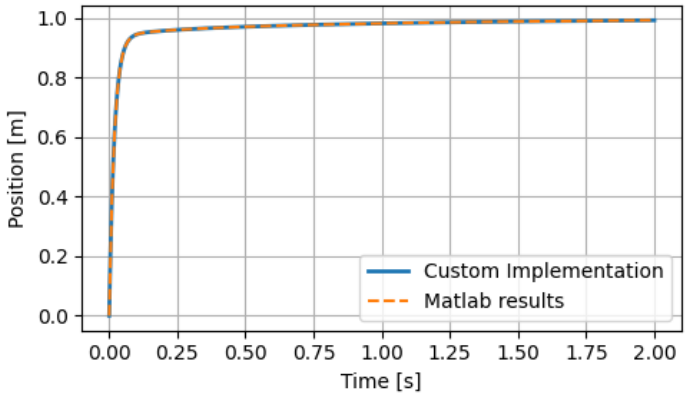


Figure 3.4: Simulation of a spring-damper system controlled by a PID controller to verify the custom PID controller implementation.

this concludes the verification of the PID controller.

3.3. TD7 verification

The custom implementation of TD7, while based on the author's original implementation [99], is implemented within the framework of Stable-Baselines3. This means that in order to verify that the model has been implemented correctly, it should be compared to the author's implementation. However, this is difficult to do, as there are many random samples of random variables involved, and since the implementations are very different. Hence, instead of comparing individual values, the performance of the algorithms was compared instead.

The environment that was used for this is the `Pendulum-v1` gymnasium environment. It was used since it is a simple and intuitively understandable environment with a continuous observation and action space environment. The author's original implementation was ran for 10 runs, with 15000 training steps. Then, the custom implementation was ran for 10 runs, with the same settings. The original author's implementation achieved a mean episode reward at the end of training of -242, while the custom implementation achieved a mean reward of -146.

Next, to validate that TD7 actually performed better than TD3 (over which it is intended to be an improvement in terms of performance [99]), the two algorithms were tested together. Again, the `Pendulum-v1` environment was used, with 15000 training steps. The TD7 agent was run with the default hyperparameters for the author's paper, with the exception of setting `learning_starts=100` (to align with TD3) and `steps_before_checkpointing=7500` (to ensure checkpointing was used from the halfway point). The TD3 agent was ran with Stable-Baselines3's default hyperparameters. The learning curves of the average of 10 runs is shown in figure 3.5. TD7 outperforms TD3, as expected. Due to the similarity of the results with the author's implementation, and confirmation of TD7's performance being better than TD3 (at least for the default set of hyperparameters), the TD7 implementation was considered verified.

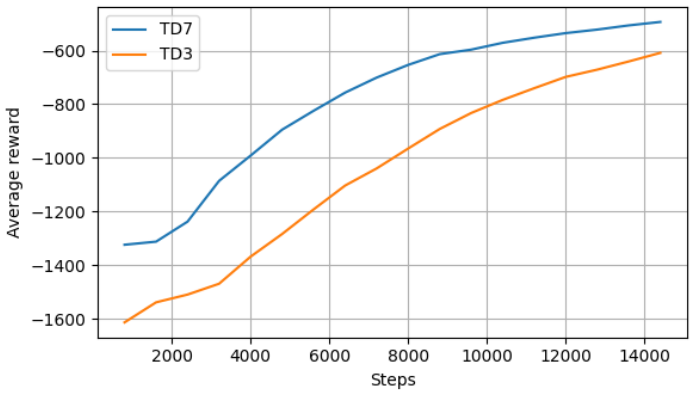


Figure 3.5: Learning curves of TD3 and TD7 for the `Pendulum-v1` gymnasium environment for 15000 steps. Note that the plotted value is the average reward per episode over the entire training process.

3.4. Rigid model verification

The spacecraft plays a central role in the full model. Hence, verification and validation efforts of the spacecraft model have been conducted in a structured manner. First, the equations of motion verification was executed, as well as a verification procedure for the implementation within the Gymnasium API. Second, the reward wrappers, which wrap around the environment, were verified by comparing it against the implementation from a different author.

3.4.1. Equations of motion

The rigid equations of motion, which are represented by Euler dynamics as discussed in equation (2.2) and the quaternion kinematics in equation (2.6), were verified by comparing it to an analytical solution, based on a chapter by Markley in the book of Wertz [142].

For the verification process, the inertia tensor was set to the axisymmetric values $J = \text{diag}\{J_1,\, J_2,\, J_3\} = \text{diag}\{J_t,\, J_t,\, J_3\}$ with $J_t = 200$ kg m^2 and $J_3 = 300$ kg m^2. Furthermore, the control torque is set to 0 in the verification process of the dynamics. The equations can then be written in scalar form as:

$$\begin{aligned} J_1\dot{\omega}_1 &= \omega_3\omega_2(J_2 - J_3) \\ J_2\dot{\omega}_2 &= \omega_1\omega_3(J_3 - J_1) \\ J_3\dot{\omega}_3 &= \omega_1\omega_2(J_1 - J_2) \end{aligned} \tag{3.4}$$

Because $J_1 = J_2 = J_t = 200$ Nm, $\dot{\omega}_3 = 0$. Next, ω_n is defined as:

$$\omega_n = \omega_3 \frac{J_t - J_3}{J_t} \tag{3.5}$$

which results in:

$$\begin{aligned} \dot{\omega_1} &= \omega_2\omega_n \\ \dot{\omega_2} &= -\omega_1\omega_n \end{aligned} \tag{3.6}$$

This describes a harmonic oscillator, and is dependent on initial conditions $\boldsymbol{\omega_0} = \begin{bmatrix} 0.05,\, 0,\, 0.01 \end{bmatrix}$ rad/s. Solving the differential equations gives the analytical solutions:

$$\begin{aligned} \omega_1(t) &= \omega_{0,1}\cos(\omega_n t) + \omega_{0,2}\sin\omega_n t \\ \omega_2(t) &= \omega_{0,2}\cos(\omega_n t) - \omega_{0,1}\sin\omega_n t \\ \omega_3(t) &= \omega_{0,3} \end{aligned} \tag{3.7}$$

Furthermore, following Markley's derivation in the book, an analytical solution for the quaternion state

can be obtained using:

$$\begin{aligned} q_1 &= h_{0,1}\cos(\alpha)\sin(\beta) + h_{0,2}\sin(\alpha)\sin(\beta) \\ q_2 &= h_{0,2}\cos(\alpha)\sin(\beta) - h_{0,1}\sin(\alpha)\sin(\beta) \\ q_3 &= h_{0,3}\cos(\alpha)\sin(\beta) + \sin(\alpha)\cos(\beta) \\ q_4 &= \cos(\alpha)\cos(\beta) - h_{0,3}\sin(\alpha)\sin(\beta) \end{aligned} \tag{3.8}$$

with:

$$\alpha = \frac{1}{2}\omega_n t \tag{3.9}$$

$$\beta = \frac{1}{2}\omega_i t \tag{3.10}$$

$$\boldsymbol{h} = \frac{\boldsymbol{L}_0}{\|\boldsymbol{L}_0\|} \tag{3.11}$$

$$\boldsymbol{L}_0 = J\boldsymbol{\omega}_0 \tag{3.12}$$

where $\omega_i = \|\boldsymbol{L}_0\|/J_t$. Note that the quaternion described in equation (3.8) is the same as the error quaternion described in equation (2.9) if the target is $\boldsymbol{q}_t = [0,\ 0,\ 0,\ 1]$.

The analytical solutions of equation (3.7) and equation (3.8) were compared to the rigid spacecraft model, with the same inertia of $J = [200,\ 200,\ 300]$ kgm^2, $\boldsymbol{\omega}_0 = [0.05,\ 0,\ 0.01]$ rad/s, and $\boldsymbol{q}_0 = [0,\ 0,\ 0,\ 1]$. The results are shown in figure 3.6. Finally, the implementation within the Gymnasium and

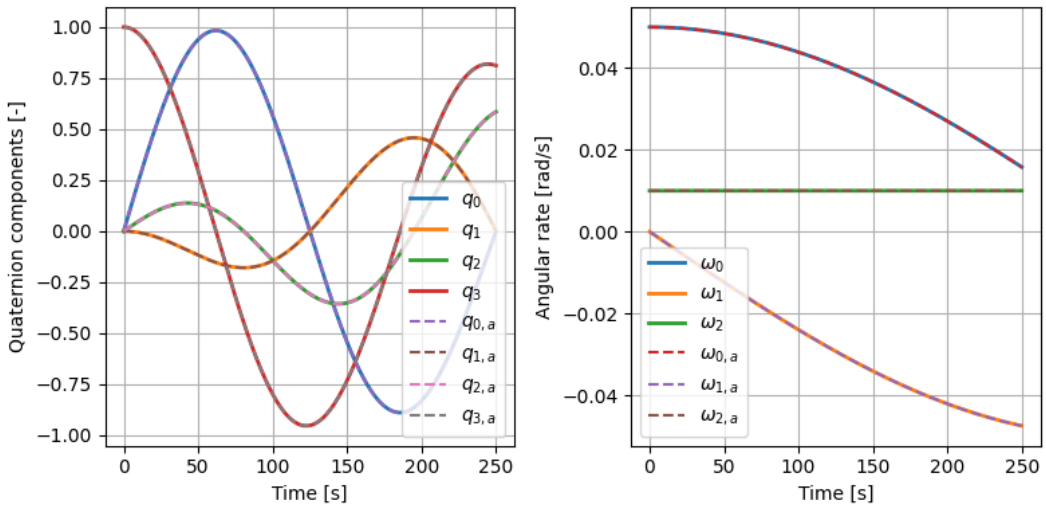


Figure 3.6: Comparison of the rigid spacecraft dynamics and kinematics with an analytical solution (subscript a) from Markley in the book of Wertz [142].

Stable-Baselines3 API was verified by using functions within those libraries. Gymnasium provides a function for this, `gymnasium.util.env_checker.check_env()`, as does Stable-Baselines3 (which uses a variant of the Gymnasium environments), `stable_baselines3.common.env_checker.check_env()`, which checks a Stable-Baselines3 `VecEnv`. The implementation of the spacecraft model was checked using these functions, and returned without errors. This verifies the implementation of the library's APIs.

3.4.2. Reward wrapper verification

As discussed previously, the spacecraft models are wrapped in a reward wrapper, which calculates the instantaneous reward. These also have to be verified, to ensure that the RL agents receive the intended reward.

The underlying spacecraft model calculates a base reward, as defined in equation (2.34). This function is only dependent on the quaternion attitude, and not on the rate. The implementation of this base

reward was done by setting the state to zero rates and a given quaternion, for which the reward can be analytically calculated using equation (2.34). The results of this are listed in table 3.1.

Table 3.1: Test quaternions and their corresponding expected reward, together with the actual reward given by the spacecraft model.

Quaternion	Expected reward	Reward	Difference
$[0, 0, 0, 1]$	1.0	1.0	0
$[0, \sqrt{2}/2, 0, \sqrt{2}/2]$	0.167677	0.167677	0

The reward wrapper implements the reward function discussed in section 2.3.2. Since this form of the reward function was proposed by Elkins, the reward function of equation (2.30) was compared to the publicly available implementation of Elkins[1] [93], with the reward function coefficients set to $c_a = 0.5$, $c_p = 1$, and $c_r = 0.25°$.

It should be noted that in Elkins implementation their previously discussed truncation bonus/penalty was disabled for this simulation, to simplify the comparison. Furthermore, a mistake was found in their implementation. They used an equation equivalent but different to equation (2.6), but forgot the factor $\frac{1}{2}$ in the quaternion derivative[2]. After fixing this, and reviewing that the propagated states were the same as those obtained using the (verified) equations of motion of the custom implementation for the same conditions, the reward function verification was done. The comparison of the reward functions for the same initial conditions, model parameters, numerical settings, and (random) applied actions are shown in figure 3.7.

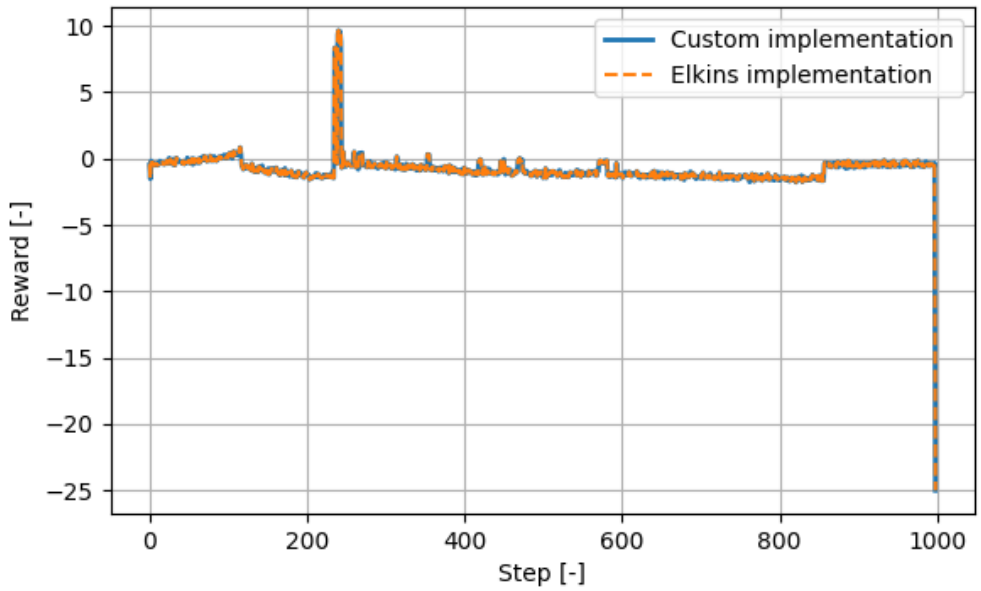


Figure 3.7: Verification of the reward function by comparing it with Elkins' implementation. The environment state is reset manually at step 234 to a state just within 0.25° from the target, resulting in a +9 bonus. After 500 steps, the episode truncates automatically, as the maximum episode steps was configured for 500 steps for this test. The reset sets the state with a rotational rate that is almost at the termination limit. Under the influence of the (same) random action, both environments reach the terminal rotational velocity at timestep 998, resulting in a penalty instantaneous reward of -25 at this final timestep.

[1] As mentioned in their paper, the code they used is available via `https://github.com/jakeelkins/rl-attitude-control`.

[2] It is not expected that their results or conclusion would significantly differ had they not made this mistake, as the dynamics and kinematics still exhibit the same behavior. It however does prevent a 1 to 1 comparison between a correct custom implementation and the results shown in their paper.

3.5. Flexible model verification

The flexible model was implemented within the library APIs the same way as the rigid model. However, since the model is not the same, it should be verified separately as well. This is done first by checking the flexible model implementation in a rigidified case, where the flexible terms are set to 0. Then, the full flexible behavior is verified against an assumed correct reference.

3.5.1. Rigidified flexible model verification

The implementation of the flexible model is first verified, again by using Stable-Baselines3's `check_env` function. After verifying that the implementation was correct, the underlying model was verified. From equation (2.14), it can be seen that in case the rigid-flexible coupling matrix δ is set to the appropriate zero matrix, then the model reduces down to the exact same model as the rigid case, equation (2.2). This was done so that the flexible modifications to the rigid model could be verified in isolation.

The flexible environment was hence ran with the same initial conditions, numerical settings, and model parameters as the rigid model, with δ set to the appropriate zero matrix (and C and K were arbitrary, since their behavior is not coupled to the rigid dynamics). The comparison between the flexible and the rigid models is shown in figure 3.8.

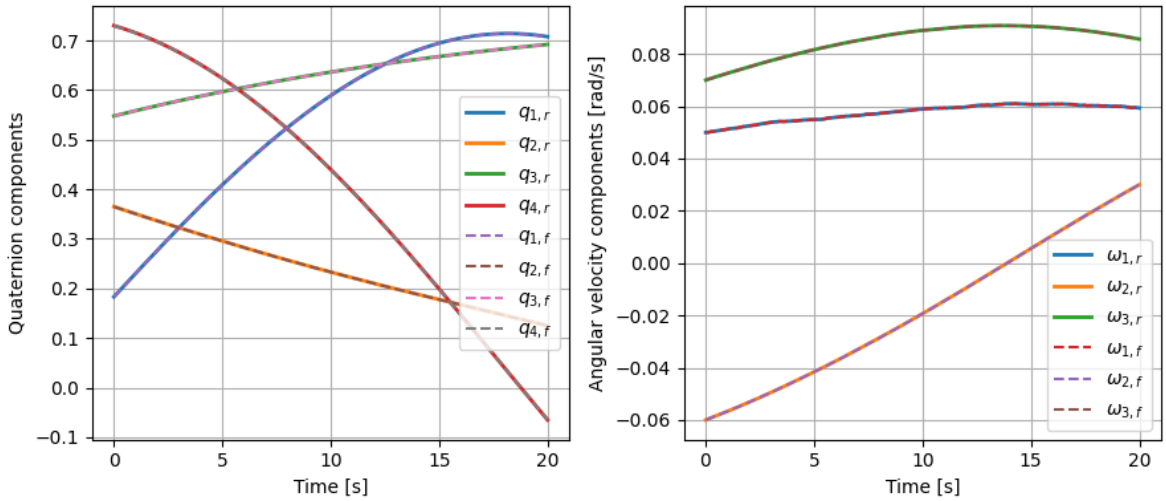


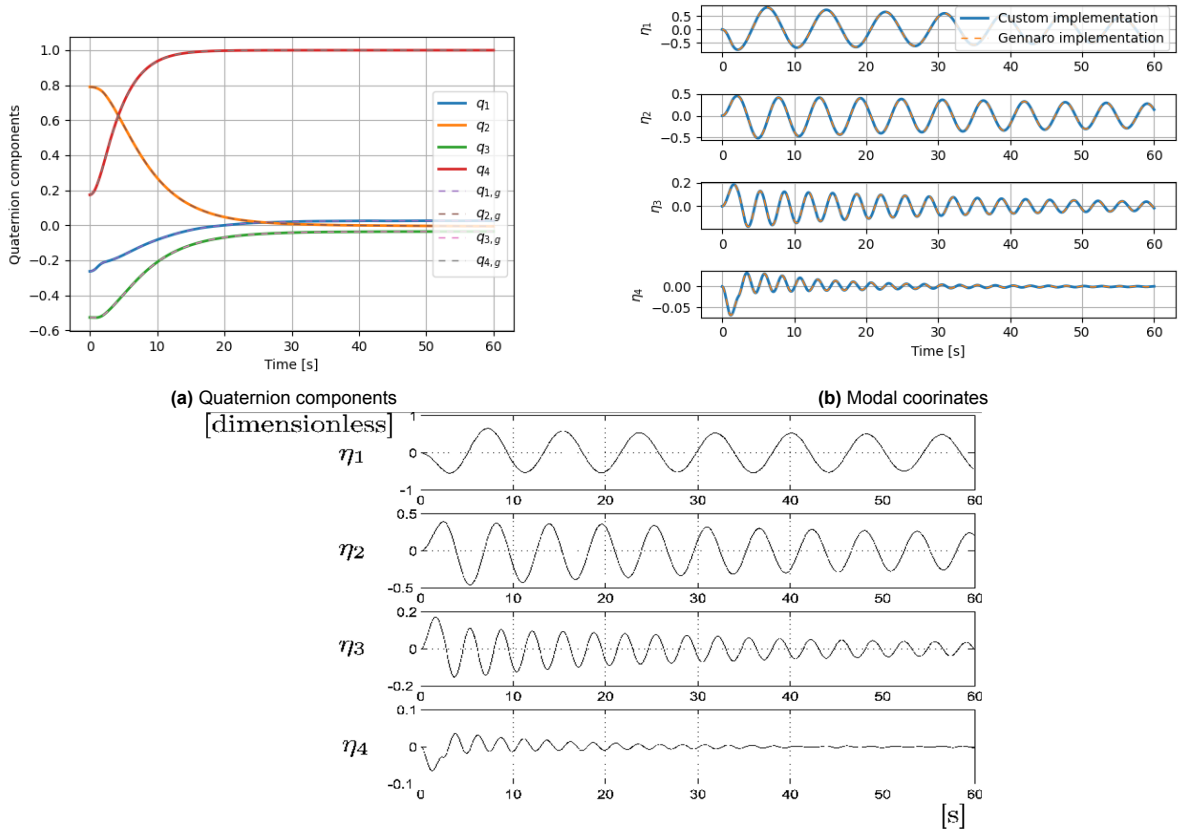
Figure 3.8: Verification of the flexible model (subscript f) with the flexible modifications to the model disabled, for the same initial conditions, model settings, and numerical settings as the rigid model (subscript r).

3.5.2. Flexible model verification

The flexible model was compared against the source of the model, namely the paper by Gennaro [116]. Their model was implemented separately from the custom flexible model implementation, and their controller was also implemented. In this section, this model is referred to as the reference model. Furthermore, the flexible model used in this research is simply referred to as the flexible model.

Since the reference model was implemented by the author of this research, instead of an off-the-shelf solution, it cannot be assumed verified. Hence, the same model parameters that Gennaro describes to have used for Figure 3 in his paper were adopted for the verification process. These parameters were inserted both in the reference model as well as the flexible model. The control torques commanded by Gennaro's controller were used for both the reference model and the flexible model. The only difference between the used settings and the ones described by Gennaro was the addition of disturbances. This is because Gennaro implemented the disturbances in a stochastic fashion, which is not exactly reproducible based on the information provided in the paper. The resulting observations of both models are shown in figure 3.9.

Note that the figures don't match with the paper exactly. This is assumed to be due to the stochastic disturbances that were not implemented. As the behavior of the results and the behavior shown by



(a) Quaternion components

(b) Modal coorinates

(c) Modal displacements from Gennaro's paper

Figure 3.9: Verification of the flexible model against the reference model by Gennaro [116]. The settings and parameters used are the same as those used to generate figure 3 of his paper. The behavior shown by these results are very similar to the behavior shown in his figures (bottom).

Gennaro's plots is very similar (for example, the characteristic first oscillation of η_4 in the first 4 seconds), the implementation of the reference model is still considered verified, and since the flexible model matches those, also the flexible model is considered verified.

3.6. Full learning process validation

The final step in the verification and validation process is the validation of the full model, which includes the learning process of the agents. This was done by comparing it against the work by Elkins [93]. His work only considers PPO and TD3. As the agents themselves are all considered verified, if the full processing pipeline (which the only component that has not been verified) is verified for one model, the full model is considered verified for all agents.

Elkins' implementation was slightly adapted, to work with newer versions of the libraries. Furthermore, the factor $\frac{1}{2}$ that was mistakenly forgotten as mentioned before was fixed. Finally, the different truncation reward was disabled.

The comparison between Elkins' work and the presented work was done using PPO. Elkins implementation is assumed validated. Hence, the method for validating the full model was to check whether the learning behavior of the full model was the same as Elkins model, irrespective of the exact parameters used. The parameters used were taken from the script by Elkins. This consisted of an inertia tensor of diag$\{0.872, 0.115, 0.797\}$ kgm^2, an integration timestep of 1/240s, impulsive control modelled by a constant torque during one time step then 20 time steps with 0 applied torque, a rate limit of 0.5 rad/s, a maximum control torque of 0.5 Nm, a maximum of 500 steps per episode, a training time of maximum 10000 episodes, and the reward function parameters $c_a = 0.5$, $c_p = 1$, and $c_r = 0.25°$. The hyper-

parameters were the default for PPO from Stable-Baselines3, with the exception of a `ReLU` activation function and a neural network architecture of two hidden layers with 400 and 300 nodes in the first and second hidden layers, respectively, for both the actors and the value functions.

In figure 3.10, two training processes completed using both Elkins' script as well as the full model of this research are shown. The learning behavior seen for this particular set of settings and hyperparameters is that the agent quickly learns how to get positive rewards instead of negative rewards, but it fails to get the state to within the requirements (set to 0.25° as per Elkins). Hence, it does not get the $+9$ bonus of equation (2.28). After the initial rapid learning process, performance deteriorates. While results from individual runs can differ slightly, especially increasing in variance at a later stage in the training, This behavior is identical for both Elkins implementation as well as the implementation used in this research. Hence, the full model of this research is considered validated.

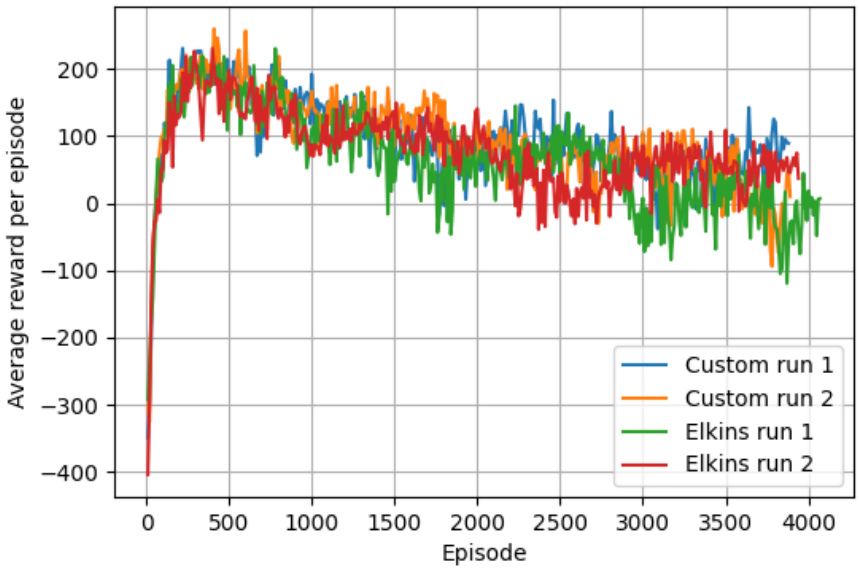


Figure 3.10: Validation process of the full model: comparison of the learning behavior between the custom implementation used in this research and the implementation by Elkins, for two distinct random seeds.

4

Results and Discussion

In this chapter, the primary results of the different experiments are shown and explained. This is done in a structured manner following from the research methodology.

Due to the extent of the experiments done, not all results can be shown and discussed, and as such, only results deemed relevant are listed here. All other results are still included, but are shown in appendix A.

Key to this chapter is that when it is stated that an algorithm is able to produce an agent with certain properties, what is meant is that at a certain training step, the usage of the discussed algorithm has resulted in at least one agent that has that certain property.

First, the results are only discussed for the rigid case, as this case is simpler to analyse. This is broken down in several sections. In section 4.1, the baseline scenarios for the rigid case were investigated and compared in multiple ways. Next, in section 4.2 the influence of different variants on the reward function are compared. Then, the effects of the domain randomizations are discussed in section 4.3. Section 4.4 then discusses the influence of hyperparameters on the performance of the agents. Finally, the effects of the flexible uncertainties are investigated in section 4.5. This section includes an in-depth discussion of the nuances of the flexible case, as well as a repeat of the analysis of the previous topics. However, for brevity and ease of interpretation, only the the differences with the rigid case are highlighted and discussed.

4.1. Baseline agent comparisons

Within this section the hyperparameters mentioned in table 2.2 were used. The discussion of the results of this baseline configuration is divided into several subsections. First, the behavior of the controllers for a single episode is compared in section 4.1.1. Then, the learning processes are compared in section 4.1.2. Next, a comparison with the reference PD controller is discussed in section 4.1.3. This is followed by a discussion of the differences in compute time in section 4.1.4, and is concluded by a discussion of the robustness of these baseline agents in section 4.1.5.

The discussion of this section about the rigid baseline case provides the fundamental background and reference performance for the more special cases in the later sections. Those cases, including the flexible case and perturbed experiments, are where the research questions are answered. This section also serves to give the reader an understanding on how to interpret the figures shown in later sections.

4.1.1. Agent control behavior

The four algorithms were trained with their baseline settings on the rigid environment. The best performing agents were saved during the training process. The behavior of these best performing agents is assessed in this section.

For a random initial state of $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, the controller

behavior of the best performing PPO, SAC, TD3, and TD7 agents respectively have been shown in figure 4.1, figure 4.2, figure 4.3, and figure 4.4.

From these figures, it can be seen that all agents settle within the requirements. PPO is the slowest, then TD3, then SAC, and TD7 is the fastest to settle. Furthermore, from these results, PPO seems to saturate the control torque and alternate between the minimum and maximum torque more readily than the other agents, which might contribute to the longer settling time.

When looking at the attitude rates, PPO seems the most aggressive, as it reaches the highest (absolute) attitude rate component of the four algorithms (roughly 10 deg/s compared to 5.5, 4, and 4.5 deg/s for SAC, TD3, and TD7 respectively).

Finally, when looking at the instantaneous reward (top right subfigures), a direct conclusion about agent behavior is difficult to draw. However, PPO seems to be the only algorithm that once it obtains the +9 requirement bonus loses it again for a short time (in this case between 158 and 181 s and between 194 and 206 s). Based on the Euler axis rotation error (top middle subfigure of figure 4.1) it seems that this is caused by an overshoot. For the other algorithms, overshooting is not an issue.

4.1.2. Agent learning comparison

The worst and best case behavior of the agents is analysed first. A couple of interesting observations can be identified from the data which are visualised in figure 4.5. All algorithms are able to produce an agent that is able to bring the final state of at least one evaluation episode to within the requirements (visible in the min/max bound shading). Furthermore, all algorithms are able to produce agents that have a convergence rate of 100%.

The first differences between the agents becomes apparent when looking at settling time (top middle), reward (bottom middle), and total control effort (bottom left). All agents except for PPO are able to produce agents that perform better than the PD controller, meaning with a lower settling time, lower average control effort, or higher reward (although these are not necessarily the same agents).

The implication of these observations is that each algorithm except for PPO has the capability to produce agents with performance metrics that are compliant with the requirements or even exceed the performance of the PD controller with the rigid spacecraft model at some point during the training process. In order to circumvent possible performance degradation later during training, simply, the best performing agent during training is saved. The question of reliability of these agents is assessed later.

Furthermore, it can be seen that all agents, except for SAC, have trouble with reliably converging. Only SAC after step 163000 has a consistent convergence rate (top right) of 100%. While TD7 near the end of its training has an average convergence rate of 95%, TD3 and PPO fare worse with convergence rates of approximately 76% and less than 50% respectively. To further illustrate this difference between individual agents, the learning curve for TD3 and PPO are shown for individual agents (each with a random seed) in figure 4.6.

To get a total episode reward higher than 500 (the maximum of equation (2.29) on its own), the agent needs to be within the requirements at least once during an episode, in order to get the $+9$ instantaneous reward bonus of equation (2.30). Both algorithms demonstrate the capability to do this almost consistently at least once during the episode, as illustrated by figure 4.6. However, convergence is defined as whether the last time step of the episode was within the requirements (see section 2.4), and this is more difficult to achieve consistently. This can be seen in the high variance in the final angle subfigure (top). Another important note is that most PPO runs demonstrate relatively consistent improvement during learning in the first 400000 steps, as shown by the consistent decrease in final angle, after which for most agents policy collapse takes place.

Since the training process is not always reliable, it is difficult to talk about an agent's performance in general terms. Hence, in the next sections, a filtering process is applied to the results. For filtered results, only runs that result in a 100% convergence rate for at least one evaluation during the learning process are considered. The performance of agents are then discussed in general terms using these filtered results.

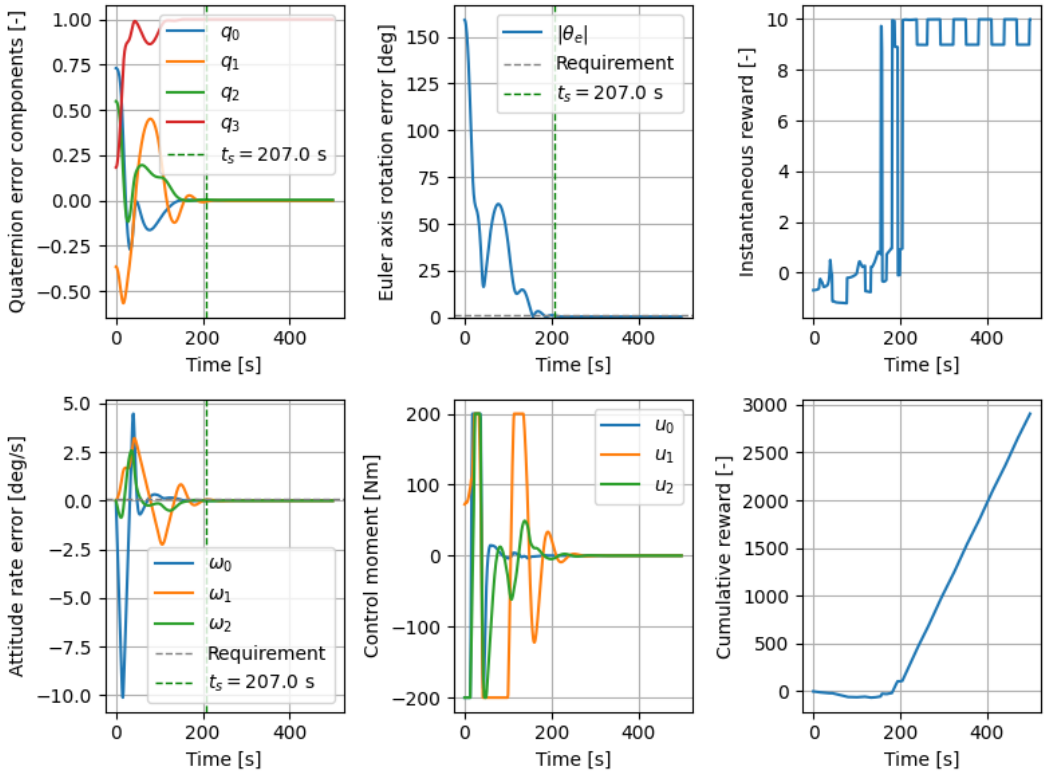


Figure 4.1: A single episode for the rigid model, controlled using the best PPO agent. Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

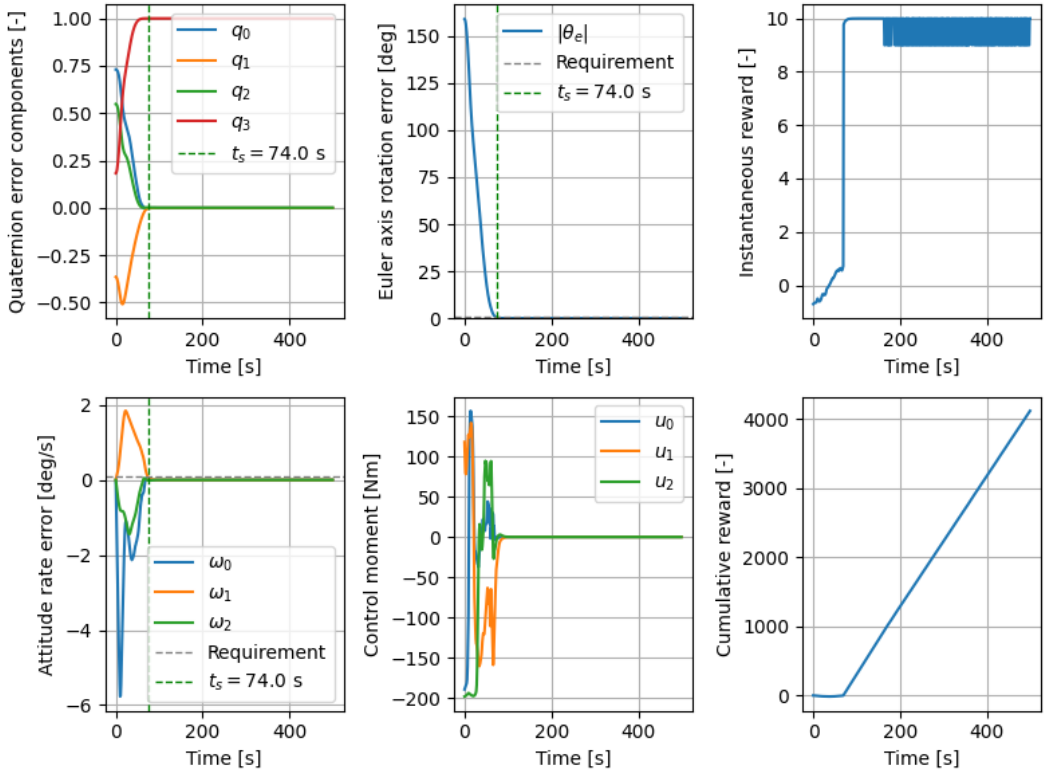


Figure 4.2: A single episode for the rigid model, controlled using the best SAC agent. Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

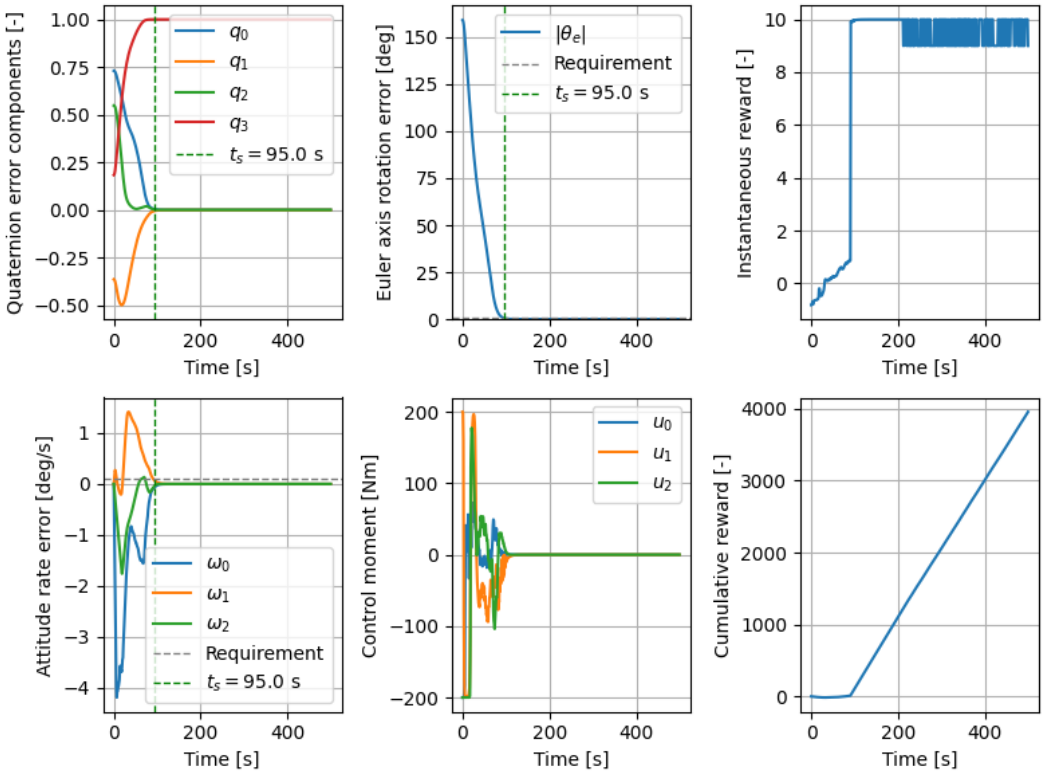


Figure 4.3: A single episode for the rigid model, controlled using the best TD3 agent. Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

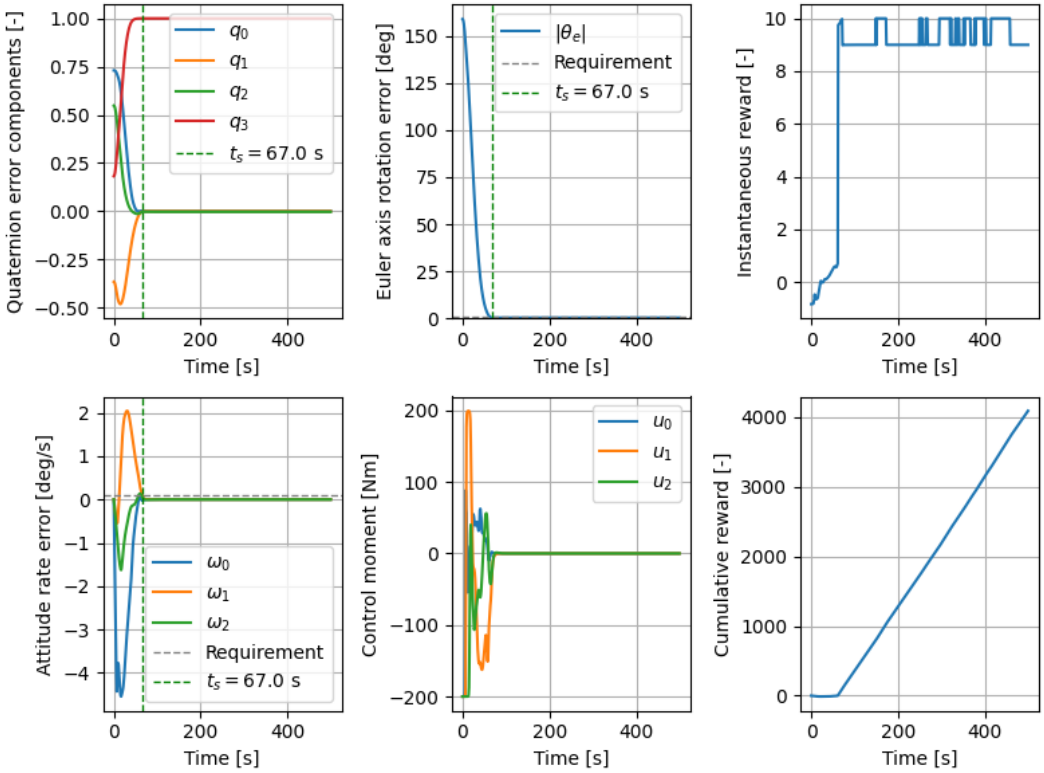


Figure 4.4: A single episode for the rigid model, controlled using the best TD7 agent. Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

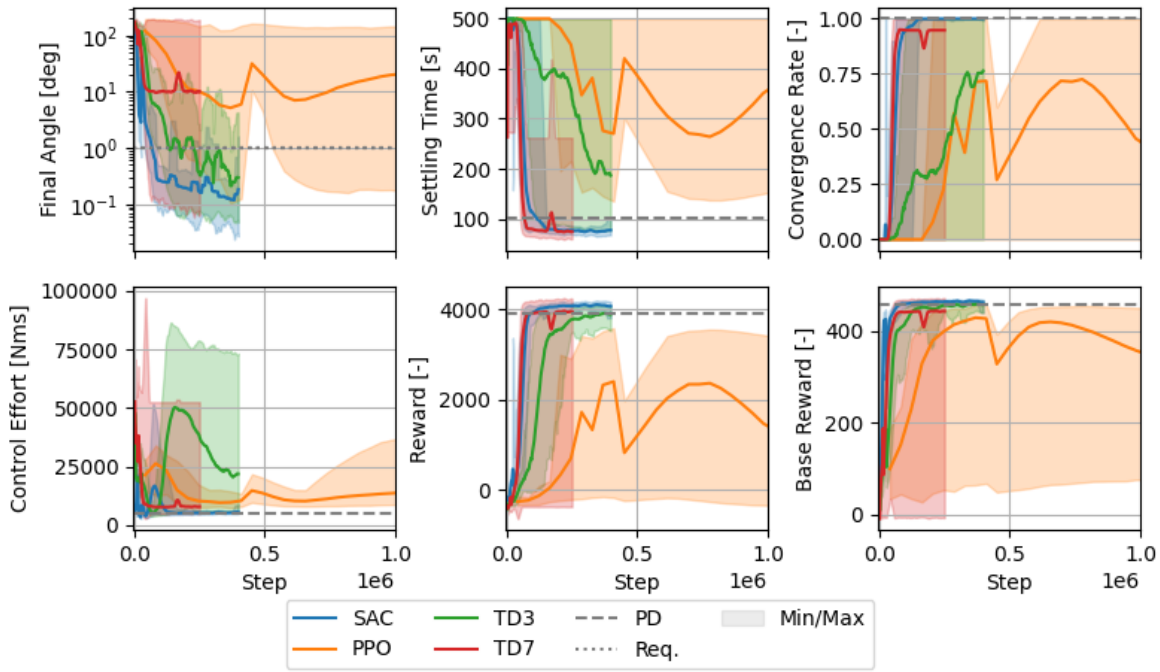
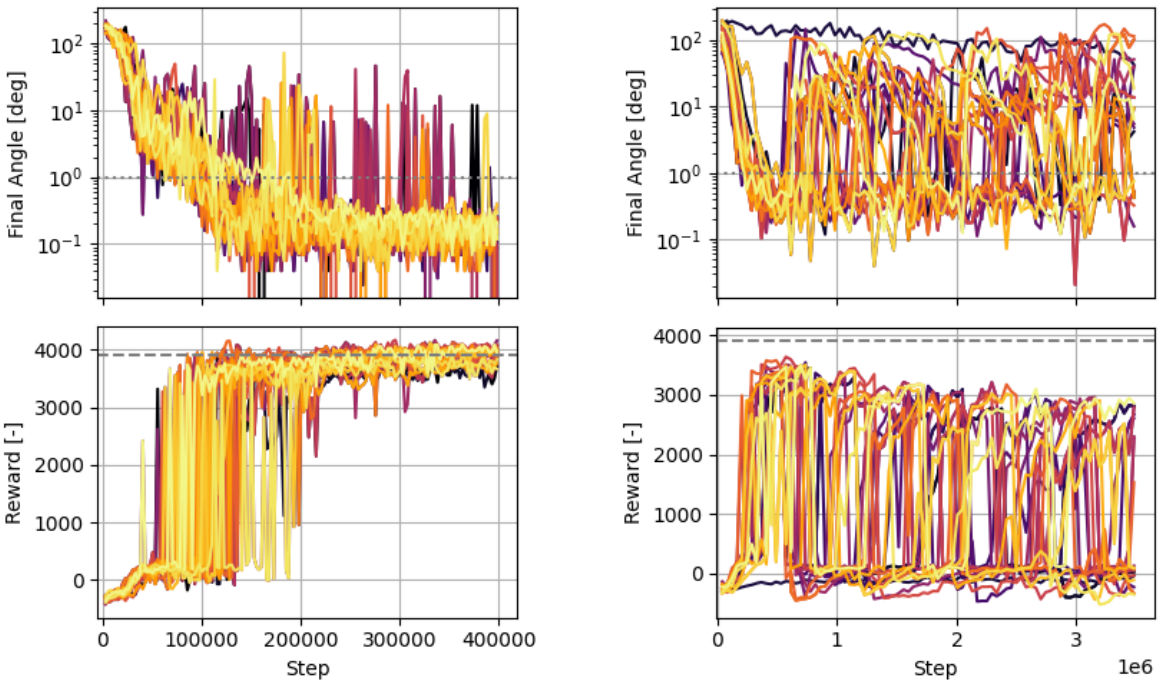


Figure 4.5: Average, minimum, and maximum results for multiple learning processes with the four different algorithms in the rigid case, with 10-step window smoothing applied.



(a) TD3

(b) PPO

Figure 4.6: Individual runs of the different algorithms for the rigid baseline case without smoothing applied. Different colours represent different runs, while the gray horizontal striped line represents the total episode reward achieved on average when using the rigid PD controller.

4.1.3. PD comparison

As a result of the filtering, the results of figure 4.5 change slightly. The new filtered results are shown in figure 4.7.

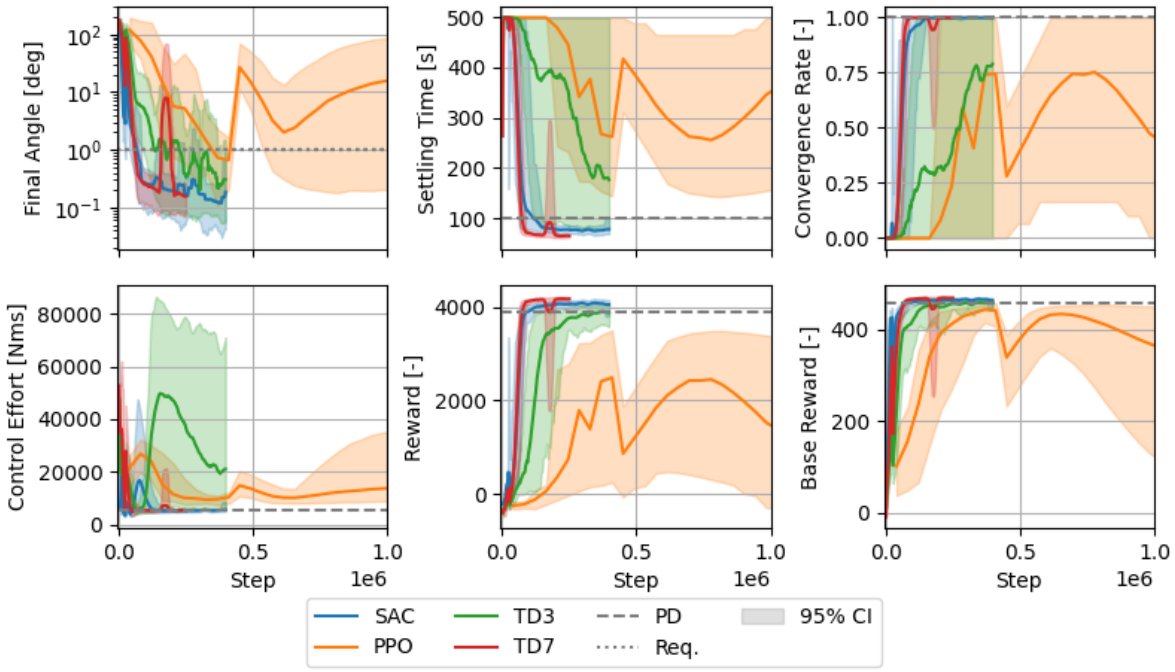


Figure 4.7: Results of the learning process of the four different algorithms in the rigid spacecraft environment. Average results are shown including a 95% confidence interval, with 10-step window smoothing applied.

In this rigid case, the performance of the algorithms can be summarized as follows, when looking at the settling time (top middle subfigure): The PPO algorithm can be dismissed when comparing to the PD controller, as it consistently underperforms. TD3 sometimes shows performance better than the PD controller, but on average is worse. SAC is the only that consistently outperforms the PD controller, while also learning quite quickly.

In terms of final angles, SAC shows the best performance. However, when settling time and episode reward are considered, TD7 shows the highest performance, conditional on agents that perform within the pointing requirements.

4.1.4. Compute time results

For RL algorithms, performance is not the full picture. Even though only the initial training is considered here, and no online fine-tuning as explained in section 1.6, looking at computational time gives a sense of the learning speed, sample efficiency, the computational cost, and potential training cost of the agents.

Two types of computational times are considered. The first is wall-time, where the actual time for the training to complete is measured, as well as a measure of the average amount of training steps per second. The second is a metric of how fast it achieves a good performance, measured by the average amount of training steps (and, using the average steps per second) until an agent receives an average episode reward that exceeds that of the PD controller. Note that the timestep at which an agent reaches this performance level for the first time is used, even if the agent's performance degrades later during training. As PPO does not exceed the performance of the PD controller, also the reward at 100k steps is shown, so that its sample efficiency can still be compared. For the rigid baseline case, the computational wall-time cost can be found in table 4.1, and the comparative compute time metrics can be found in table 4.2.

Table 4.1: Computational time metrics for the rigid baseline case.

Agent	Steps per second	Training time [s] (total steps)	Time per 100k steps [s]
PPO	324.9	11262 $\pm$ 2176 (3.5M)	307.8
SAC	55.0	7346 $\pm$ 710 (400k)	1819
TD3	59.2	6873 $\pm$ 952 (400k)	1688
TD7	20.7	12543 $\pm$ 2756 (250k)	4826

Table 4.2: Comparative computational time metrics for the rigid baseline case.

Agent	PD exceed steps	PD exceed time [s]	Reward at 100k steps
PPO	-	-	-226.9 $\pm$ 39.1
SAC	59000 $\pm$ 1073	1073	3996.5 $\pm$ 53.8
TD3	184977 $\pm$ 3112	3122	1647.8 $\pm$ 1679.7
TD7	52949 $\pm$ 2555	2555	4142.3 $\pm$ 17.7

From table 4.1, it can be seen that PPO executes the most training steps per second. However, the reward at 100k steps is still very low (as seen in table 4.2), in comparison to the other algorithms, indicating a much lower sample efficiency. It is also clear that TD3 is the fastest off-policy algorithm in terms of steps per second, followed not far behind by SAC, and that TD7 is the slowest of all.

When looking deeper into the results from table 4.2, this view changes slightly. Even though TD3 goes through iterations the fastest of all the off-policy algorithms, it is the slowest in reaching the performance of the PD controller. Furthermore, TD7, even though it is much slower in terms of steps per second than TD3, reaches the PD level performance sooner than TD3, on average.

A couple of interesting remarks can be drawn from these results. First, if the performance of PPO could be stabilized and policy collapse could be prevented, PPO could be a very strong candidate for an online learning agent. Second, in case computational constraints are not limiting and in cases where sample efficiency is the most important, then TD7 seems promising, as it gets the highest reward at an equal number of training steps of all algorithms, with the lowest variance at that timestep as well. Third, SAC's training process seems to be most balanced, with relatively fast training speed, good sample efficiency (with relatively low variance), and relatively good reliability (when combined with the insights from figure 4.7, as TD7 shows at least in one case a short-lived policy collapse at around step 200000).

4.1.5. Baseline robustness analysis

As explained in chapter 2, the robustness is assessed by applying several perturbations, one by one, to each agent. The performance metrics of the baseline agents when perturbed can be seen in table 4.4, where the number next to the performance metric (second column) indicates the index of the pertubation applied, as listed in table 4.3. This table is shown in full here (instead of in the appendix) as it provides the important reference values for the sensitivity analyses in the following sections. The PD controller performance is highlighted in orange, while agent performances equal to or better than the PD controller have been highlighted in blue, to aid visual comparison. Note that this table is unfiltered, so it displays the average results of the agents both at the evaluation with the highest reward (best agents) and at the end of training (final agents).

A few things stand out from this table:

1. There is a large difference between the performance of the best agents and the performance of the final agents, except for that of SAC. This indicates that the learning process is unstable to some extent for the agents. However, this does not meaningfully change in the perturbed cases (indices 2-10).
2. None of the agents converge for any of the perturbations in the gyroscope (indices 4-6). This indicates that the policy of the agents is not generalized enough to be robust to these perturbations. In many ADCS architectures, a state estimator is included to deal with this issue, which could potentially also improve performance here, but is not within the scope of the current research. An alternative could be to let the agent learn online, but as discussed before, this is also outside of

Table 4.3: Keys for the perturbations listed in table 4.4.

Perturbation	Perturbation index
Unperturbed	1
Inertia scaling	2
Inertia rotation	3
Gyro noise	4
Gyro constant bias	5
Gyro drift	6
Torque misalignment	7
Torque scaling	8
Torque noise	9
Disturbance torque	10

the scope of the current research. It is clear that the algorithms are not able to deal with these types of perturbations by their own.

3. All agents have difficulty with the torque scaling perturbation (index 8). This is a similar type of problem that is sometimes resolved using for example a state estimator (or in this case more appropriately, a disturbance observer) or other methods. The algorithms are also not sufficiently able to deal with these types of perturbations on their own.
4. Inertia scaling (index 2) seems to have little effect on the performance of the controllers. With respect to this perturbation, the agents are hence already robust.
5. Inertia rotation (index 3) seems to be manageable for the baseline agents. Reliability is slightly degraded with convergence rates of less than 100% in some cases, and a higher average settling time and total control effort. However, the other performance metrics don't degrade significantly. The agents are hence already relatively robust to this perturbation, but not fully.
6. The performance values for the remaining torque perturbations (misalignment, noise, and the disturbance torque, with indices 7, 9 and 10 respectively) don't differ much from the unperturbed case. However, notable differences can still be identified, for example how for torque misalignment, even though the mean episode reward is the same as the unperturbed case, the settling time is significantly larger. Moreover, for torque noise and the disturbance torque, the only notable difference is visible in the total control effort. This shows that the agents are able to deal with these perturbations very effectively already, at the cost of some extra control effort.
7. When comparing the PD controllers with the agents in terms of mean total control effort (regardless of the perturbation applied), the agents perform slightly or significantly worse, as visible by a lack of blue shaded cells. This means that in the rigid case, even if both the PD controller and the agents are robust to a perturbation, the PD controller generally achieves a lower total control effort. Hence, the agents never offer an advantage when looking at this performance criterion.

It should be noted that comparisons in later sections with agents that failed to learn properly (such as all baseline agents in the gyro perturbations) can't be used to discuss the learning performance. Hence, these are omitted from discussion in the following sections.

4.2. Effects of reward function variations

The first variation to be investigated in detail is the design decision of the reward function, equation (2.30). Here, the coefficients and their default values that were changed are $c_r = 1°$, $c_a = 0.5$, and $c_p = 1$. The full reward function is repeated here:

$$r_{a,t} = \begin{cases} \exp\left(\frac{-\phi_t}{0.14 \cdot 2\pi}\right) - c_a \frac{\|\boldsymbol{M}_t\|}{\|\boldsymbol{M}_{\max}\|} & \phi_t \leq \phi_{t-1} \\ \exp\left(\frac{-\phi_t}{0.14 \cdot 2\pi}\right) - c_a \frac{\|\boldsymbol{M}_t\|}{\|\boldsymbol{M}_{\max}\|} - c_p & \text{otherwise} \end{cases} \tag{2.29}$$

$$r_t = \begin{cases} r_{a,t} + 9 & \text{if } \phi_t \leq {c_r}^\circ \\ r_{a,t} & \text{otherwise} \end{cases} \tag{2.30}$$

Table 4.4: Rigid robustness performance of the baseline agents. Shown are mean performance values of the best performing agents during training (best) or the agents at the end of training (final), before filtering for convergence. Highlighted in blue are performance values that are equal to or better than the PD controller (right-most column). Note, for the convergence rate and mean episode reward, this means that the values are higher than the PD, but for mean settling time, control effort, and final angle, the values are lower than the PD.

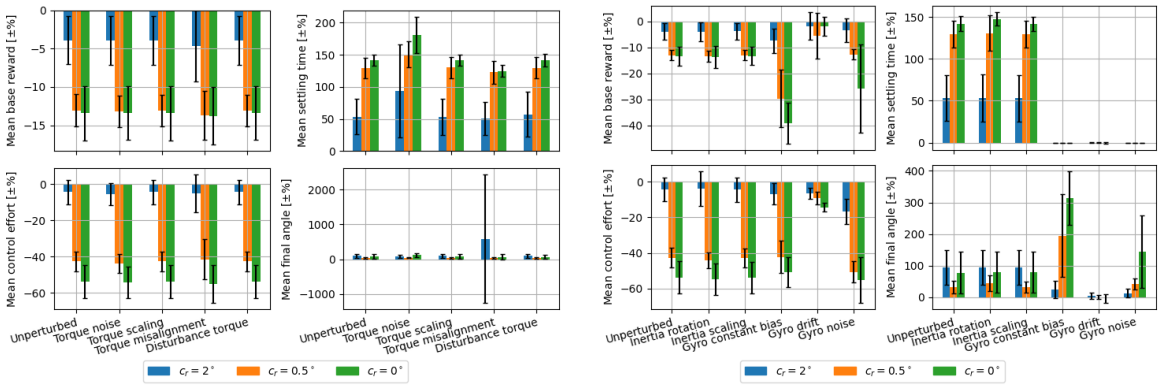
		Best agents				Final agents				
	Index	PPO	SAC	TD3	TD7	PPO	SAC	TD3	TD7	PD
Convergence rate	1	1.00	1.00	0.99	1.00	0.30	1.00	0.77	0.95	1.00
	2	1.00	1.00	0.99	1.00	0.30	1.00	0.77	0.95	1.00
	3	1.00	1.00	0.82	1.00	0.30	0.84	0.48	0.95	1.00
	4	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0	0
	7	1.00	1.00	0.99	1.00	0.30	1.00	0.80	0.95	1.00
	8	1.00	1.00	0.95	1.00	0.30	1.00	0.75	0.95	1.00
	9	1.00	1.00	0.96	1.00	0.30	1.00	0.77	0.95	1.00
	10	1.00	1.00	0.99	1.00	0.30	1.00	0.77	0.95	1.00
Mean episode reward	1	3350.4	4063.4	3889.7	4156.5	733	4050.7	3860.1	3894.4	3942.2
	2	3348.8	4052.1	3887.7	4160.8	731.9	4051.4	3860.7	3896.5	3943.5
	3	3330.4	4057.5	3827.8	4148.7	717.7	4010.5	3761.3	3902.3	3940
	4	-219.3	-383.3	-429.3	-484.1	-247.4	-383.4	-422.9	-476.2	-314.8
	5	115.2	77.5	15.6	172.3	-13.3	101.7	-27.5	169.8	46.6
	6	-624.6	-579.9	-593.5	-607	-607.4	-574.8	-579.1	-598.3	-569.9
	7	3252.5	3994.6	3827.9	4116.9	675.9	3993.2	3809.5	3871.5	3877.9
	8	3348.3	4057.8	3873.9	4155.5	728	4049.2	3852.4	3898.2	3932.8
	9	3343.5	4038.1	3863.7	4133.9	721.3	4034	3853.9	3900.1	3896.8
	10	3347.9	4044.4	3878.7	4132.8	730.2	4046.7	3856.1	3903	3942.2
Mean settling time	1	157.9	79.8	98.4	69.4	416.4	80.1	187.8	78.2	97.5
	2	158	79.8	98.4	69.4	418.8	80.1	188	78.3	97.4
	3	160.7	79.9	169.7	69.8	415.9	148.8	305.8	78.6	98.6
	4	499	499	499	499	493	497.8	499	486.5	499
	5	499	499	499	499	491.8	499	499	486.5	499
	6	498.6	442.4	447.1	462.4	493.8	436.5	436.9	453.9	429.9
	7	169.2	85.4	129.2	73.2	424	85.2	202.3	82.2	105.6
	8	158.4	79.8	116	69.5	417.9	80.1	195	78.2	98.4
	9	159	79.9	146.9	69.5	423.7	80.2	190.7	78.4	97.6
	10	157.9	79.8	98.4	69.4	416.3	80.1	187.8	78.2	97.3
Mean control effort	1	10730	5790	8673	5867	20829	5877	19853	8374	4917
	2	10744	5790	8505	5870	20847	5875	20029	8391	4901
	3	11035	5975	21418	6341	21783	14979	39122	9051	5073
	4	72730	90081	94976	94126	54748	90098	96449	91570	90662
	5	10858	6211	18118	6295	20888	6244	17439	8621	5723
	6	90554	85971	88649	91099	86806	84955	87107	89898	84583
	7	11261	6521	9610	6327	22926	6305	19374	8783	5704
	8	10751	5805	9762	5888	21095	5885	21129	8369	4915
	9	11245	7373	11182	7537	21648	7584	21506	10083	5554
	10	10763	5844	8703	5924	21183	5930	19895	8421	4966
Mean final angle	1	0.4	0.1	1	0.2	22.9	0.1	0.2	10.5	0
	2	0.4	0.1	1	0.2	27	0.1	0.2	6.9	0
	3	0.4	0.1	1	0.2	26.7	0.1	0.2	12.6	0
	4	13	36.6	45.1	67	46.7	37.2	44.5	83.6	20.8
	5	8	16.8	19.2	13.3	25.8	16.2	22.3	22.8	24.2
	6	171.7	192.6	178.1	172.5	183	180.4	176.5	167.5	186.6
	7	0.4	0.9	1.3	0.2	30	0.1	0.2	9.5	0
	8	0.4	0.1	1	0.2	27.1	0.1	0.2	11.3	0
	9	0.4	0.1	1	0.2	25	0.1	0.2	10.1	0.1
	10	0.4	0.1	1	0.2	24.1	0.1	0.2	11	0

All the results were be compared with the baseline scenario, in which the default values were used for the reward function coefficients. The sensitivity of the mean episode base reward, mean settling time, mean total control effort, and mean final angle were assessed for PPO, SAC, TD3, and TD7 separately. Note that the base reward of equation (2.34) and not the actual reward (which is used for the domain randomized agents and the hyperparameter tuned agents to be discussed later) was used because the actual reward function changes, which as a result cannot be compared 1 to 1. The sensitivity of these metrics is shown by the percentage change in these metrics with respect to a baseline controller, as discussed in section 2.5.3. The sensitivities are also shown for perturbed cases, where the sensitivity is with respect to the baseline controller under the influence of the same perturbation.

The best performing agents were considered, not the agents at the end of the training process. Furthermore, only the agents that upon evaluation had a 100% convergence rate for the unperturbed case were used for the calculation of the performance measures, meaning that the data was filtered.

The first parameter to be changed was the coefficient governing the $+9$ requirement bonus. The default value, $c_r = 1°$, was changed to $c_r = 2°$, $c_r = 0.5°$, and $c_r = 0°$. By doubling and halving the default value, a sense of the sensibility of the agents to this value was obtained. Furthermore, by disabling the bonus altogether (by setting it to zero), a qualitative insight on the effect of the coefficient was obtained.

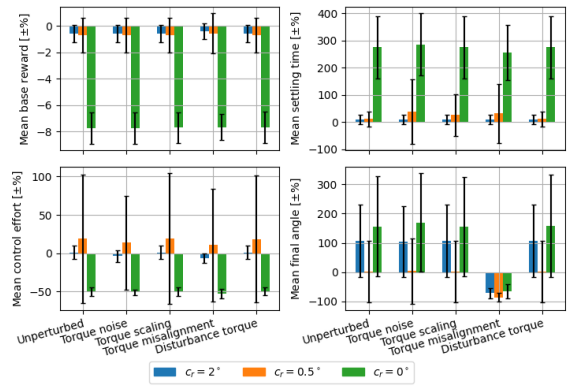
The results for PPO, SAC, TD3, and TD7 can be found respectively in figure 4.8, figure 4.9, figure 4.10, figure 4.11. In these (and other sensitivity figures later), the error bars represent the standard deviation, as discussed in section 2.5.3.



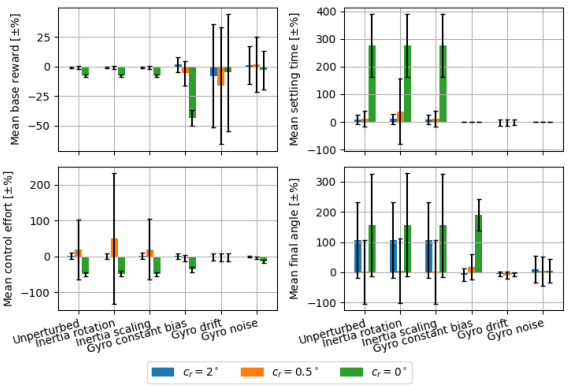
(a) PPO - Torque perturbations

(b) PPO - Inertial and gyroscope perturbations

Figure 4.8: Performance of PPO agents trained with variations in c_r in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.

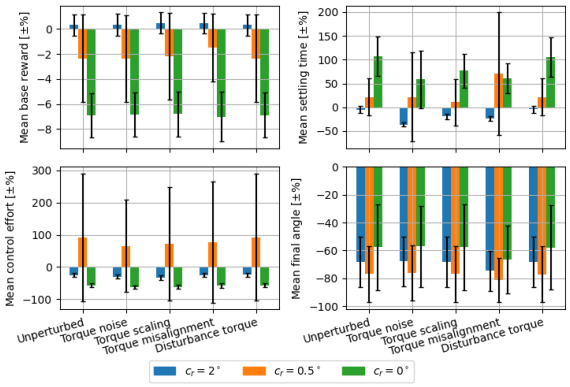


(a) SAC - Torque perturbations

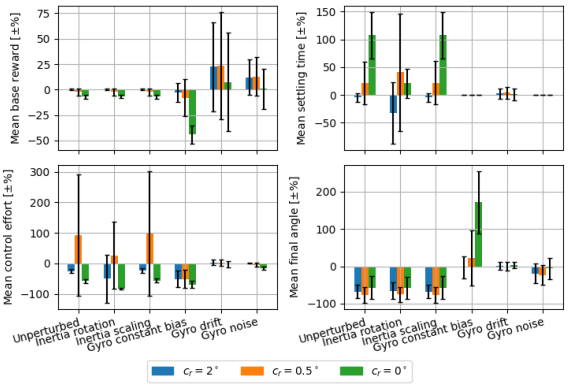


(b) SAC - Inertial and gyroscope perturbations

Figure 4.9: Performance of SAC agents trained with variations in c_r in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.

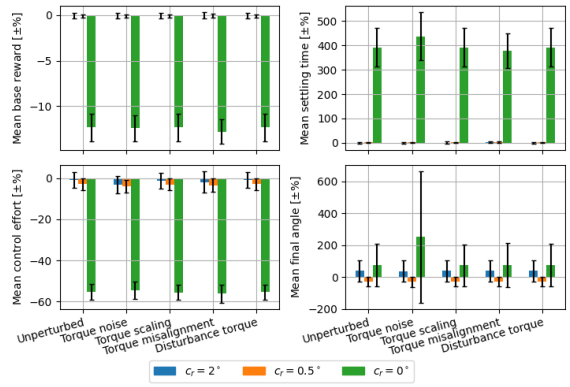


(a) TD3 - Torque perturbations

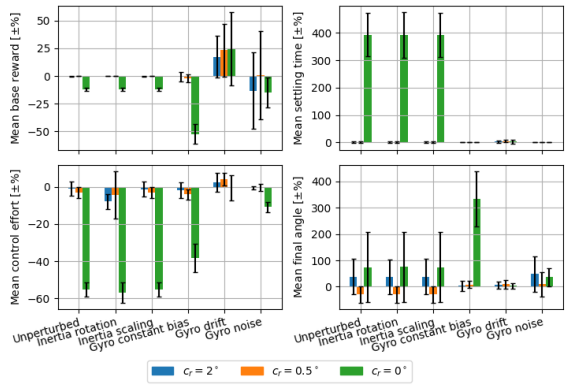


(b) TD3 - Inertial and gyroscope perturbations

Figure 4.10: Performance of TD3 agents trained with variations in c_r in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.



(a) TD7 - Torque perturbations



(b) TD7 - Inertial and gyroscope perturbations

Figure 4.11: Performance of TD7 agents trained with variations in c_r in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.

When looking purely at the unperturbed cases, the influence of a non-zero c_r is clear. For PPO, any

change in c_r degrades the controller performance, in terms of base reward and settling time. This means that for PPO, $c_r = 1^\circ$ is (close to) a local maximum in unperturbed performance. For the off-policy algorithms however, the story slightly changes. Setting $c_r = 0^\circ$ seems to degrade base reward performance by a few percent, and also seems to be detrimental to the settling time, with extreme cases seeing a 400% increase in settling time. Hence, it is clear that the inclusion bonus reward for states within the requirements pushes the agents to achieve the target state much faster, with a small penalty of extra required control effort over the entire episode, as seen in the bottom left subfigures for $c_r = 0^\circ$. A higher value for c_r seems to be beneficial only for TD3, where reward increases and settling time decreases when $c_r = 2^\circ$. SAC and TD7 are not significantly affected by an increase or decrease of c_r.

The robustness to the torque perturbations (except torque scaling) and both types of inertia perturbations is not affected significantly by a change in c_r. This is clear from the similar changes in performance for the unperturbed case. This makes sense for the rigid case, as the system is deterministic. Since these perturbations should have a mean zero effect, all they do in the rigid case is add noise to the environment experience sampling for the agent. More importantly, the requirement bonus should not affect the ability of an agent to generalize directly.

It should be noted that the sensitivity for the gyro perturbations is less intuitive to interpret, as all RL agents for the default c_r had a convergence rate of 0 for those perturbations (see table 4.4). In general, robustness to a torque scaling, random drift in the gyro bias or random noise in the gyro might be affected by a change in c_r, but the spread of the data (standard deviations are on the order of or larger than the averages) is too high to draw any such conclusions with the sample size.

The standard deviation of the results for the second coefficient c_a is generally larger than the change in the mean performance. Furthermore, the percent change is on the order of 1% for PPO and on the order of 0.1% for the off-policy algorithms. Hence, the effect of this change on the base reward is not significant. The results for this have been included in appendix A.

The most significant effect of c_a would be expected in mean total control effort, as c_a directly penalizes a higher instantaneous commanded control torque. When c_a was disabled (no penalty) by setting $c_a = 0$, the mean control effort for every algorithm indeed increased significantly. However, also the variance in the data rises dramatically. This is explained by the fact that c_a imposes a soft constraint on the control torque directly. When it is removed, an agent is free to choose any control torque, without a direct effect on the instantaneous reward. Hence, c_a might be acting as a regularizer on the control torque. However, as stated before, the variance of the data is too larget to confidently draw such conclusions.

The final coefficient, c_p, was the final coefficient to be investigated. This coefficient penalizes the agent in case the new state is not better than the previous state. The idea is that this forces the agent to climb the gradient in the reward function as quickly as possible for the greatest reward.

The results from these tests show that the variance is also too large to draw any reliable conclusions. The variance in base reward, settling time, and mean control effort is sometimes several times larger than the change in the average, for both the unperturbed and most perturbed cases. This data is still shown in appendix A.

The large increase in variance across the board without a clear relationship between c_p and the performance metrics indicate that this parameter does not provably improve the learning process. In fact, due to the increase in variance without any benefits, it reduces the consistency and hence the reliability of the learning process.

4.3. Effects of domain randomization

The first domain randomizations that were investigated were the inertia perturbations: scaling and rotation. The filtered learning curves for the former is shown in figure 4.12.

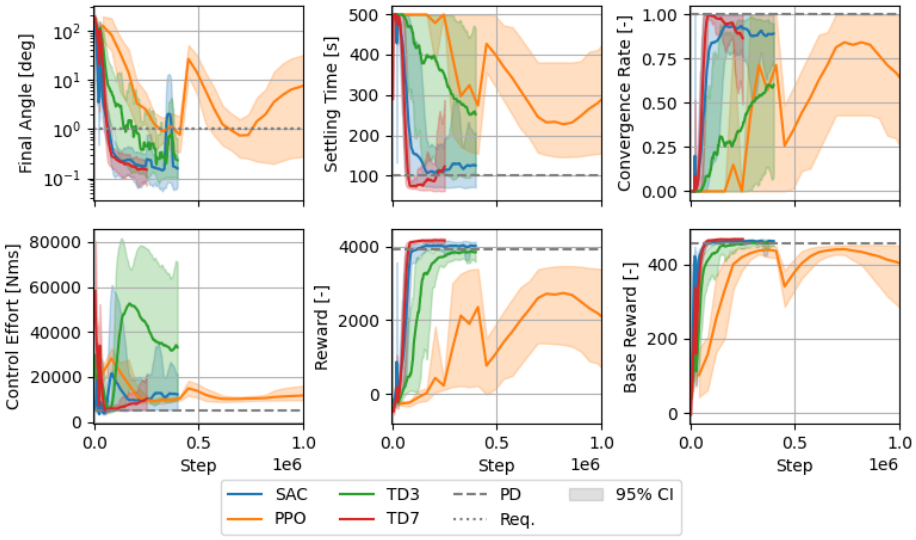


Figure 4.12: Results of the learning process of the four different algorithms in the rigid spacecraft environment, with an inertia rotation domain randomization applied. Average results are shown including a 95% confidence interval, with 10-step window smoothing applied.

In comparison with figure 4.7 (the baseline case), the learning process looks very similar. The same behaviors that were described for that case also apply for these domain randomizations, which indicates that the applied domain randomizations do not influence the agents' learning behavior in a significant way. In fact, the learning process for the majority of the domain randomized agents did not seem to change significantly, except for the domain randomizations applied to the gyroscope.

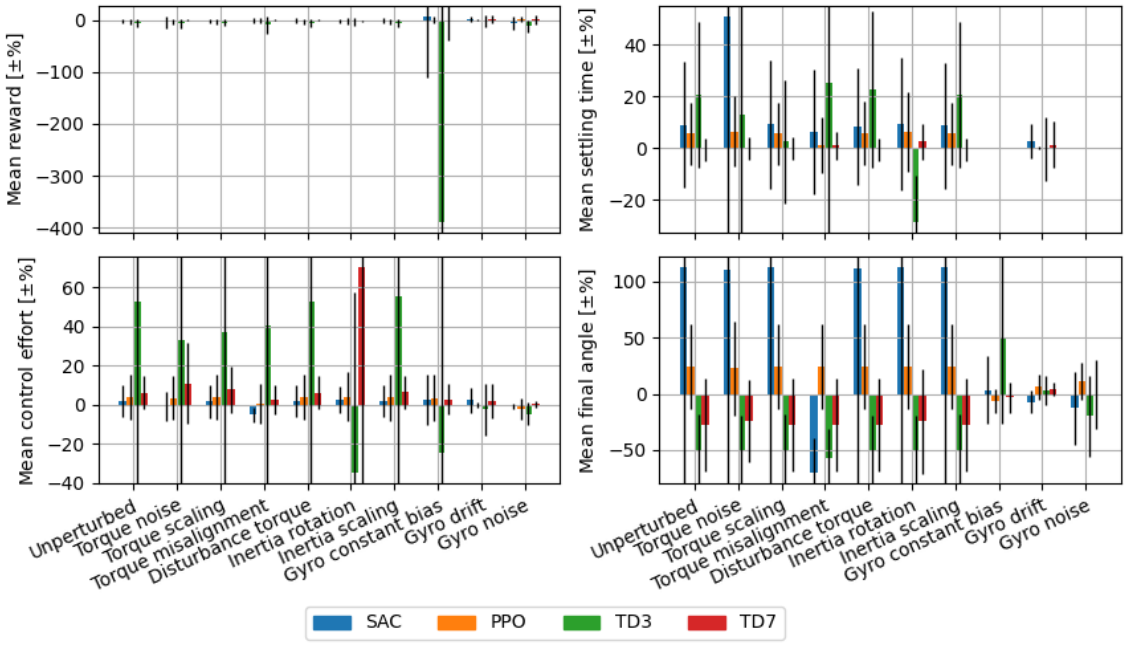


Figure 4.13: Performance of agents with domain randomization applied in the form of inertia rotation, with respect to the baseline performance. Bars indicate one standard deviation.

From the performance sensitivity for the inertia rotation domain randomized agent shown in figure 4.13, it can be seen that for the unpertubed case, the performance in terms of reward does not meaningfully change for all agents, with a worst-case average change of -5% (for TD3) and no difference for TD7

Table 4.5: Summary of performance for the PD controller and the baseline (BL) and inertia rotation domain randomized (DR) TD3 and TD7 agents. The performance shown is the performance of the controllers for the rigid case when the inertia rotation perturbation is applied.

	PD	**TD3 BL**	**TD3 DR**	**TD7 BL**	**TD7 DR**
Convergence rate	1	0.82	1	1	1
Episode Reward	3940	3827.8	3658	4148.7	4096.1
Episode base reward	458.4	452.7	448.1	465.3	464.5
Settling time [s]	98.6	169.7	121.1	69.8	71.6
Control effort [Nms]	5073.6	21418.7	13923.6	6341.5	10803.2
Final angle [deg]	0	1	0.5	0.2	0.1

in the best case. The standard deviation is also very large. However, when looking at the settling time and mean control effort statistics, TD3 shows a meaningful improvement over the baseline case for the inertia rotation perturbation, with the settling time and control effort decreasing with around 30% on average. TD3 is the algorithm that showed the highest amount of variance, while TD7's and SAC's baseline cases were already capable of producing policies that could deal with this perturbation relatively well. Nevertheless, TD3's convergence rate with the inertia rotation perturbation applied increases from 0.82 (see table 4.4; relevant numbers repeated in table 4.5) to 1.0, and its mean settling time decreases from 170 s to 121 s. In contrary, TD7's performance in the inertia rotation perturbation seems to not be affected with this domain randomization, with exception of the total control effort, which rises substantially. This indicates that this domain randomization is not beneficial in all cases. The performance changes of TD3 and TD7 have been summarized in table 4.5.

In the case of the inertia scaling perturbation, of which the sensitivity has been shown in appendix A, no such relationships can be identified. Although the performance in the unperturbed case does not differ significantly, an improvement in performance for the inertia scaling perturbation cannot be seen due to the high variance of the data. This is to be expected, as in section 4.1.5 it was established that the baseline controller was already very robust to this perturbation. Again, the detrimental effect of this domain randomization on the inertia rotation perturbation is seen. The high variance indicates that a clear relationship cannot be identified, but it does provide further evidence that domain randomization does not always deliver positive results in terms of robustness.

The next domain randomizations are those in the gyroscope, including gyroscope noise, constant gyroscope bias, and drifting gyroscope bias. These did not result in converging controllers, and hence the final agents are looked at instead. The results of the noisy gyroscope learning curves are shown in figure 4.14. Note that to show the extent of the failure, in this case the shading represents the minimum/maximum results of the agents, instead of the 95% confidence interval, and to ensure no (positive or negative) outliers get lost in the data, no smoothing is applied.

This result shows that none of the agents are able to find any acceptable policy in this randomized domain. Hence, the reinforcement learning algorithms as applied in this research are not able to deal with these types of uncertainties and perturbations. The same conclusion can be drawn for gyroscope bias, both constant and drifting, as their learning curves look very similar to that of the noisy gyroscope (figure 4.14). These figures have been put in appendix A.

The next domain randomizations that were investigated is the usage of a different attitude representation. Both Euler angles and modified rodrigues parameters were tried. These variations are not strictly a 'domain randomization', but fall into the same category of study variations. Since this variation is not expected to significantly influence the robustness of the agents, only the unperturbed case is discussed here. Furthermore, since the intent is to assess whether a different representation results in more stable learning, the final agents instead of the best agents are compared. The sensitivity results for the Euler representation are shown in figure 4.15a, while figure 4.15b shows the MRP sensitivity results. It should be noted that these results are filtered to the converged agents. The convergence rates can be found in table 4.6.

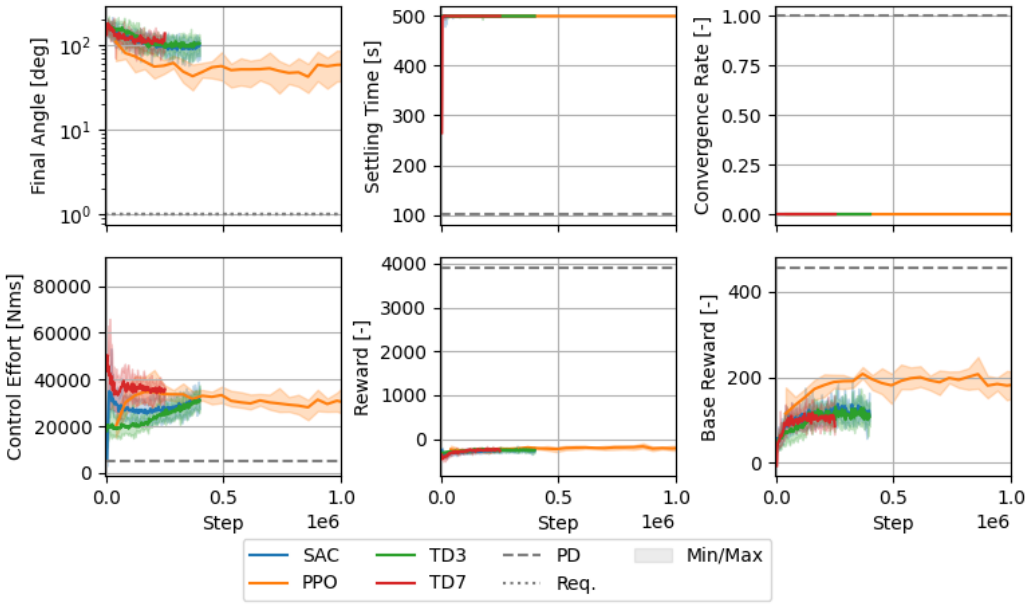
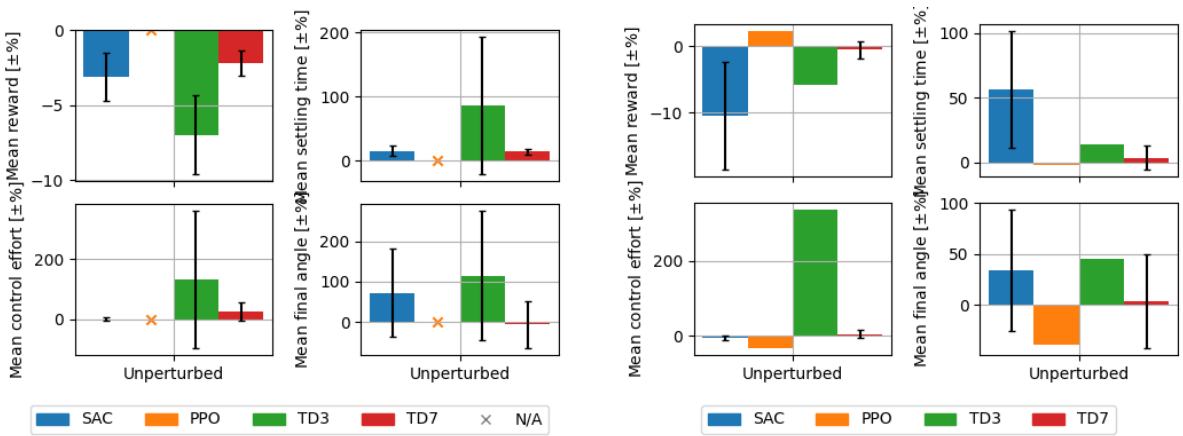


Figure 4.14: Results of the learning process of the four different algorithms in the rigid spacecraft environment, with a noisy gyroscope domain randomization applied. Average results are shown including the minimum and maximum results, without any smoothing.

Table 4.6: Average convergence rates for the final agents for each of the four algorithms in the three different attitude representation cases.

Convergence rate case	PPO	SAC	TD3	TD7
Baseline (quaternion)	0.305	1.0	0.767	0.947
Euler	0.243	1.0	0.3	1.0
MRP	0.143	1.0	0.1	1.0



(a) Euler **(b)** MRP

Figure 4.15: Performance of agents with domain randomization applied in the form different attitude representations, with respect to the baseline performance, between all agents at the end of training. The crosses indicate that no results are available due to the filtering, meaning that none of the final agents had a convergence ratio of 1.0.

It can be seen that the algorithms generally perform worse or roughly equally to the baseline. This makes sense, as the reward function is more directly related to the quaternions than the Euler angles or MRPs. Furthermore, while inherently Euler angles and MRPs give the agent the same information,

the relationship between the attitude representation and the reward function is much more non-linear than for quaternions, as the same information has to be transferred in 3 rather than 4 parameters, which could make it more difficult for the agent to learn any patterns or relationships.

TD7 and SAC seem to be the only algorithms that is not as negatively affected by any of the alternative representations as the other algorithms. Furthermore, TD7 exhibits higher convergence rates for the alternative representations. A possible explanation for why TD7 manages to retain its performance, even though the other algorithms break down to some extent, might be that TD7 spends extra computation to learning latent state-action representations [99], which are fed into the actor and critics. Still, overall, the quaternion representation performs best.

The final domain randomization parameters to be investigated are the torque randomizations (noisy, scaled, misaligned, and disturbance torques). First, the torque misalignment learning curves have been shown in figure 4.16. The learning process still looks relatively the same as figure 4.7. This means that unlike for the gyroscope domain randomizations, the agents are able to deal with this domain randomization and still result in performant policies. The same holds for the other torque domain randomizations. The sensitivity analysis for the misalignment domain randomization is shown in figure 4.17.

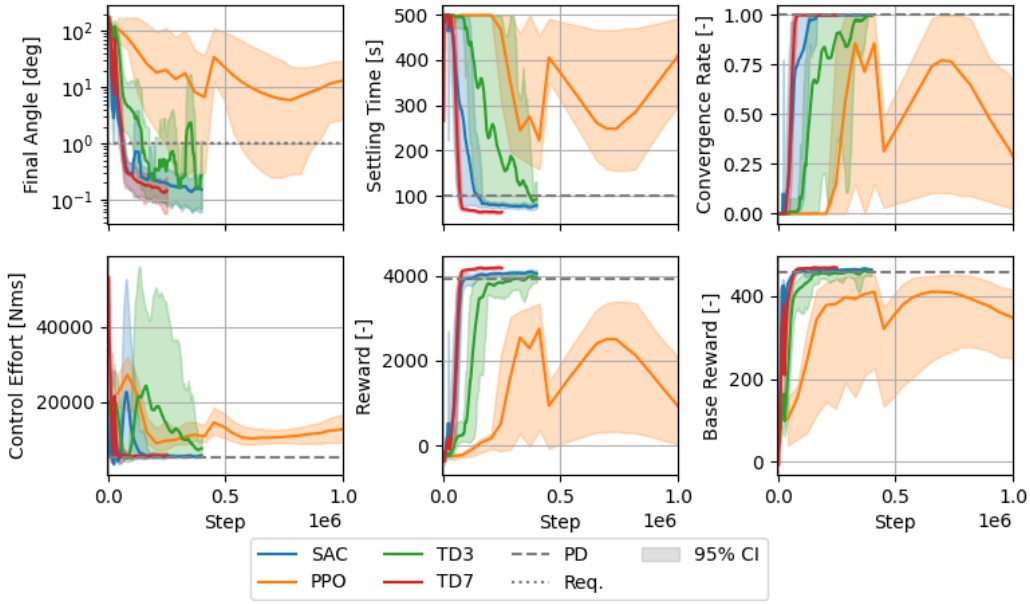


Figure 4.16: Results of the learning process of the four different algorithms in the rigid spacecraft environment, with torque misalignment domain randomization applied. Average results are shown including a 95% confidence interval, with 10-step window smoothing applied.

The standard deviation of the data is significantly larger than the change itself, which again makes it difficult to draw conclusions. Furthermore, the sensitivity to the misalignment perturbation is not different than for the unperturbed evaluations. As such, meaningful improvements in performance for the misalignment perturbation are not apparent in the sensitivity analysis.

Similar results were obtained for the torque scaling, torque noise, and disturbance torque domain randomizations. The charts showing this are displayed in appendix A. These results can be expected, as the baseline agents are already able to deal with all these torque perturbations well, as is clear from table 4.4.

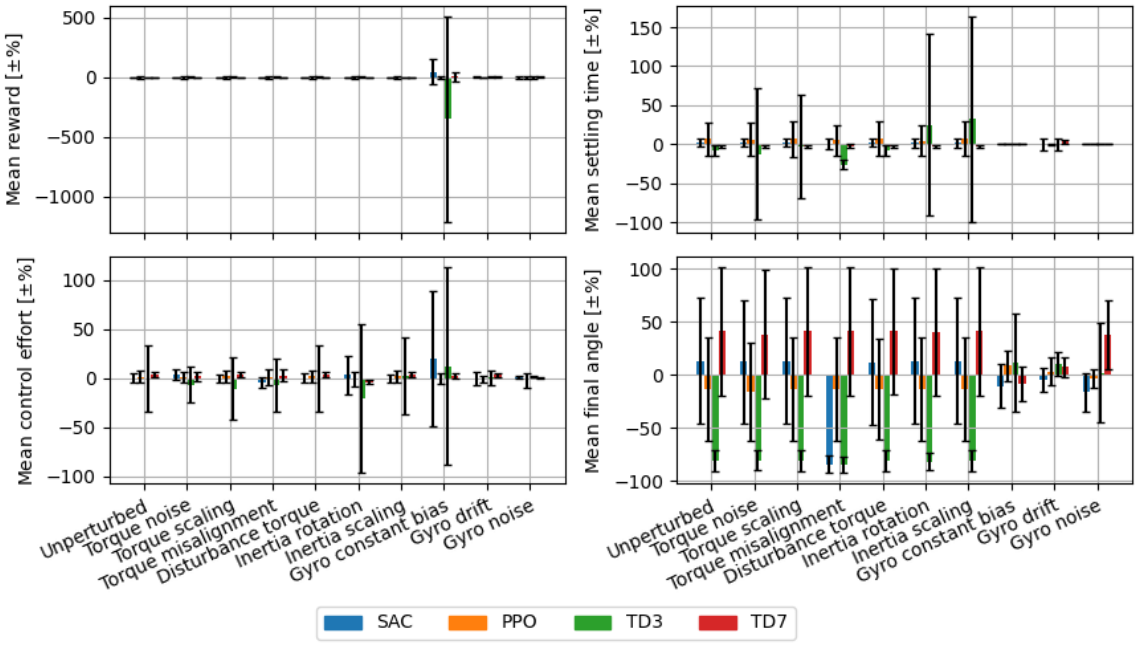


Figure 4.17: Performance of agents with domain randomization applied in the form of torque misalignment, with respect to the baseline performance.

4.4. Effects of hyperparameter tuning

The influence of the hyperparameters on the agent performance is investigated separately per hyperparameter. The goal is not to obtain the most optimal hyperparameters for the specific environment settings, but to get an idea of the relationship between the hyperparameters and agent performance. As discussed in chapter 2, the hyperparameter tuning was done automatically by Optuna. Hence, while Optuna always did multiple simulations for the default hyperparameters, the sampling of new values is done following its tree-structured Parzen estimator, semi-randomly. The domain of the hyperparameters is hence also different per algorithm (as is the default in some cases, as shown in table 2.2).

4.4.1. Common hyperparameters

The first hyperparameters to be discussed are the common hyperparameters of all agents: the exploration noise, the activation function, the discount factor γ, the learning rate, and the node count in the hidden layers. The first of these, the exploration noise, has a very similar effect on the environment interaction samples as the action noise perturbation. The only difference is at which step in the loop the noise gets applied: directly on the output of the actor, or within the environment (as described in equation (2.44)). Its performance has been plotted for the unperturbed rigid case in figure 4.18.

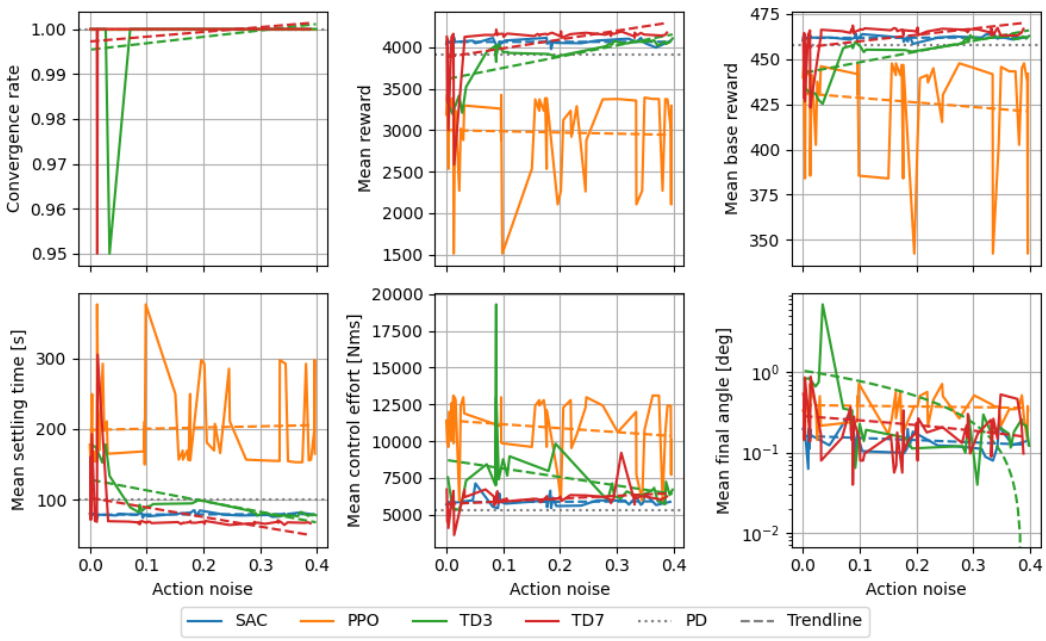


Figure 4.18: Performance of the best agents during their training process plotted against the exploration noise hyperparameter value for all four algorithms in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.55$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

From this figure, it can be seen that SAC and TD7 are not very sensitive to the action noise. TD3 and TD7 rely on it primarily to explore the state-space, and hence, a drop in convergence rate and reward can be seen for very low values of the action noise. SAC and PPO both train a stochastic policy. Since the environment is fully deterministic, a fully deterministic optimal policy exists. It makes sense that SAC and PPO therefore are less affected by extra noise included in their policies, as the gradient descent of their policies are pushed towards policies with a tighter distribution. PPO seems to perform significantly worse than the other algorithms, but not significantly worse under influence of the action noise (which is 0.0 by default for this algorithm).

The next hyperparameter is the activation function. Four options were investigated: ELU, ReLU, Sigmoid, and tanh. The results are shown in appendix A, as a clear relationship between performance of the agents and activation function cannot be identified in general. However, the convergence rate of the controllers was 100% for all tried combinations of algorithms and activation functions, meaning that all activation functions can be used for this application. When taking a more detailed look, it seems like ReLU might perform slightly better than ELU for all algorithms, and in the category of S-shaped activation functions, tanh seems to perform slightly better in terms of reward and settling time for SAC and TD3 than sigmoid, but this effect cannot be seen for TD7.

The next hyperparameter is the discount rate γ. The performance of different values for this parameter has been shown in figure 4.19. In general, values of γ above 0.985 seem to result in more variance in the results. However, higher values of γ also seem to result in a higher total reward, lower settling time, and smaller control effort (the latter with exception to TD3). For very large values ($\gamma > 0.999$), this trend seems to break down.

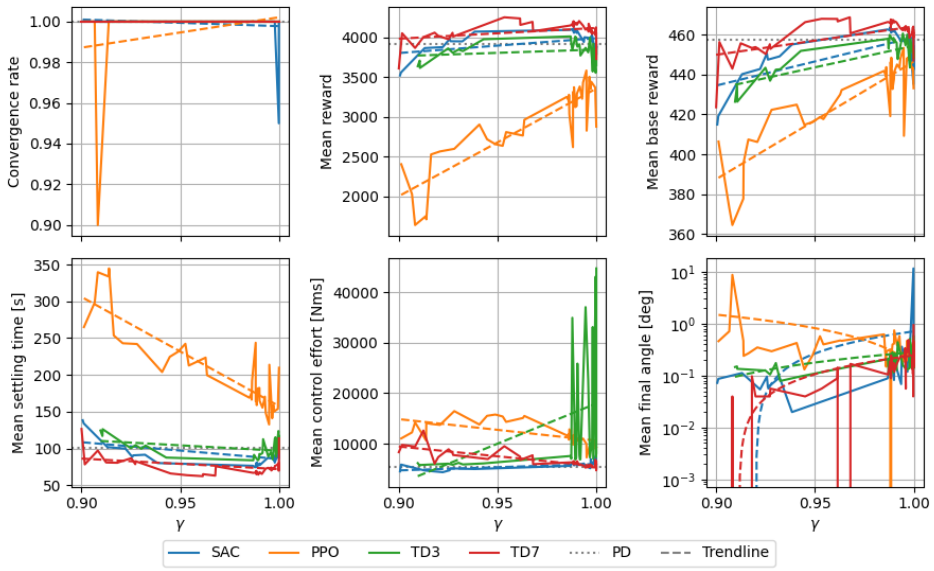


Figure 4.19: Performance of the best agents during their training process plotted against the γ hyperparameter value for all four algorithms in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.76$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

The learning rate's influence is plotted in figure 4.20. A linear relationship between its value and agent performance cannot be clearly recognized. However, for it can be seen that for SAC and TD3, a value n the order of 0.001 or lower is likely optimum, while TD7 benefits most from a value around 0.0003 (the default), and PPO has the best performance with a slightly lower learning rate. This is in disagreement with Elkins, who used lower values for TD3 (varying from 0.0003 $\rightarrow$ 0.00001).

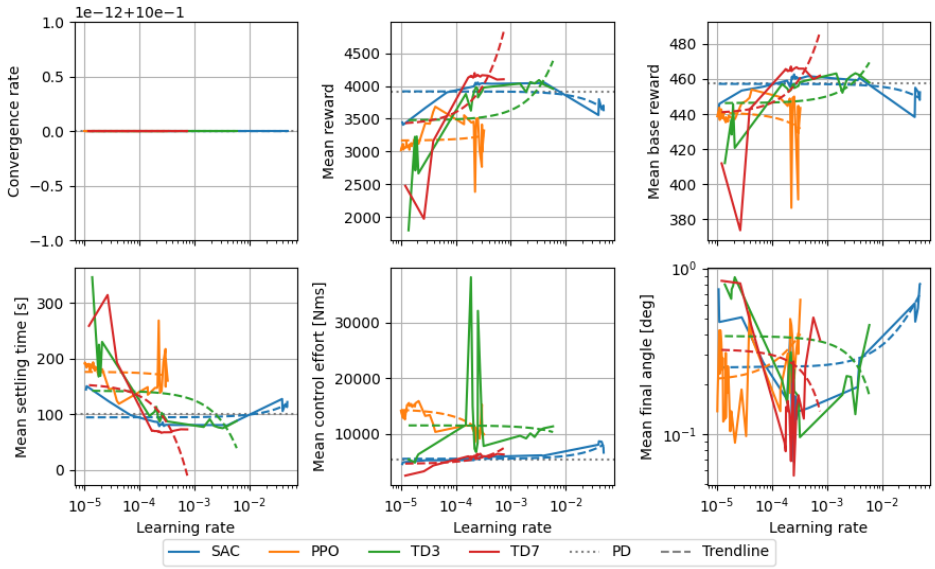


Figure 4.20: Performance of the best agents during their training process plotted against the learning rate hyperparameter value for all four algorithms in the rigid unperturbed environment. Of all the trendlines shown in the mean episode reward figure (all linear trends), the best fit is $R^2 = 0.28$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

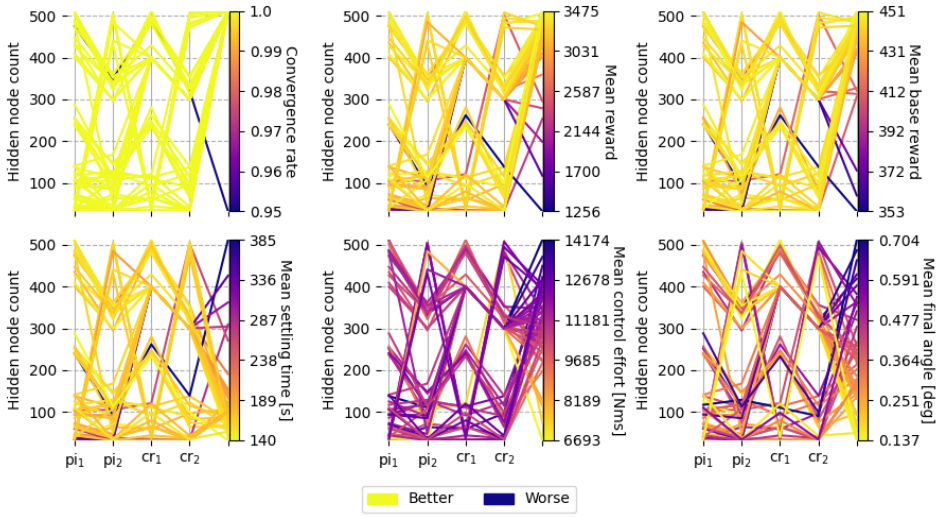


Figure 4.21: Agent performance metrics for different combinations of node count hyperparameter values for PPO, for the rigid environment.

The final common hyperparameter is the node count of the hidden layers. The performance of different combinations between the actor net (pi) and critic net (cr) hidden node counts (for two hidden layers) have been shown in figure 4.21, figure 4.22, figure 4.23, and figure 4.24.

PPO's optimal hidden node counts seem to be a little lower than its default values of 400 (first layer) and 300 (second layer). Rather, a node count of just under 300 seems to result most consistently in the highest performance. For SAC, with 256 nodes by default in both layers, the agents seem to be performing better with higher node counts, just under 500. However, the difference is not large. TD3 and TD7 both seem to benefit from node counts around this level as well, so a bit larger nets than their defaults of (400-300) and 256, respectively.

Interestingly, when looking only at the maximum control torque, the off-policy algorithms seem to benefit from lower node counts, at around 80-120. The lower node count could be having a regularizing effect, which has been previously described in works such as that of Neyshabur [143]. A smoother policy would be beneficial to the total control effort, as it would exhibit fewer characteristics of a 'bang-bang' behavior. A reason why this does not show up for PPO could be the on-policy learning strategy, which makes it less prone to overfitting irrespective of network size as it can't replay a previous experience.

4.4.2. Algorithm specific hyperparameters

First, the hyperparameters for PPO will be discussed. When looking at the best agents for the clip range hyperparameter (clipping the policy gradient, default 0.2) on first inspection seems to have a higher performance for all metrics with lower values. However, the linear fit is very bad, indicating a low confidence in the first-order relationship. Furthermore, the amount of samples for the off-default value is often low, with two agents per point. Hence, whether there is actually a performance improvement is not sufficiently certain. A similar effect shows up for the number of epochs hyperparameter. The lower values (default 10) seem to be higher performing. Although, again, the sample size is too low for the limited dataset to definitively conclude this. The figures for these have hence been placed in appendix A.

The hyperparameters GAE-λ, max gradient norm, vf coefficient, and number of steps (defaults 0.95, 2048, 0.5, and 0.5 respectively) don't show signs of affecting performance, when varying them between roughly 0.84-0.99, 512-8192, 0.2-0.8 and 0.4-0.6, respectively. The figures for these have been put in appendix A. The target KL divergence hyperparameter does not have an effect on the performance with respect to the disabled case (the default), when varied between 0.094 and 0.098. This is also put in the appendix. In conclusion, none of the hyperparameters specific to PPO seem to meaningfully affect the performance of the agents with a high degree of confidence.

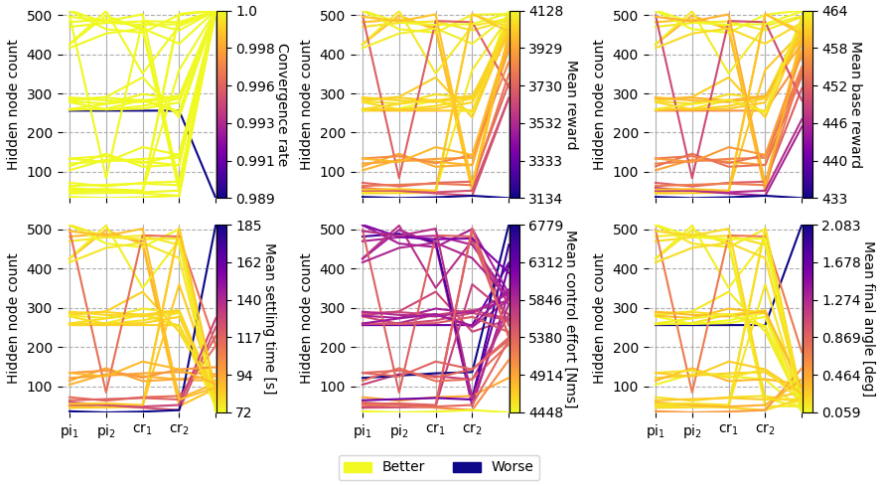


Figure 4.22: Agent performance metrics for different combinations of node count hyperparameter values for SAC, for the rigid environment.

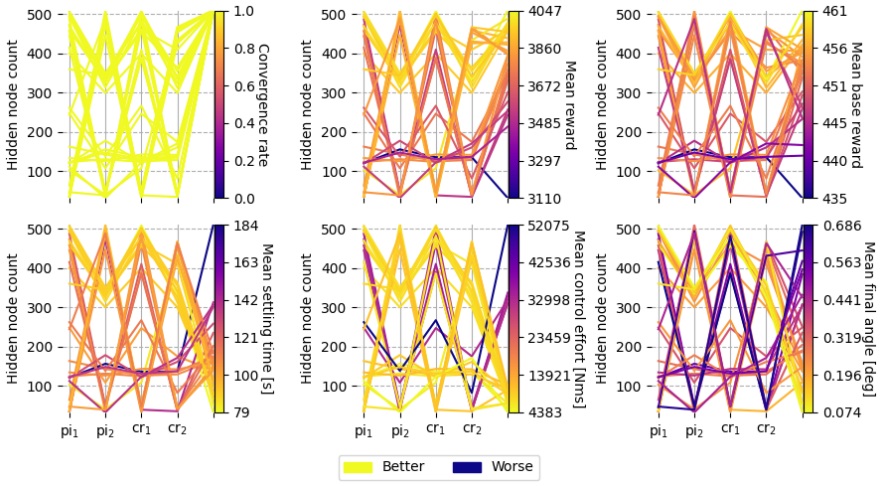


Figure 4.23: Agent performance metrics for different combinations of node count hyperparameter values for TD3, for the rigid environment.

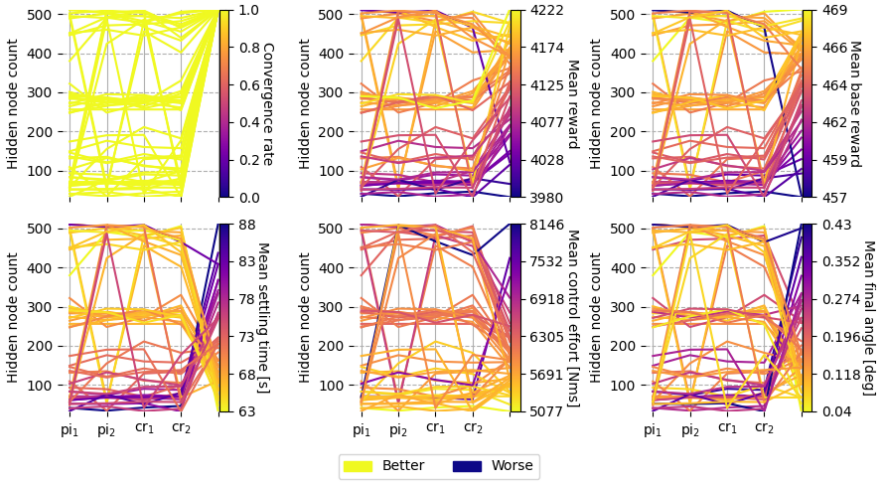


Figure 4.24: Agent performance metrics for different combinations of node count hyperparameter values for TD7 (excluding encoder layers), for the rigid environment.

Next to be discussed is SAC. Its only two non-generic hyperparameters are the target update interval and τ, the Polyak averaging coefficient. The first of these is shown in appendix A. Any change in this parameter (from the default 1) does not significantly and consistently affect the results by a meaningful amount.

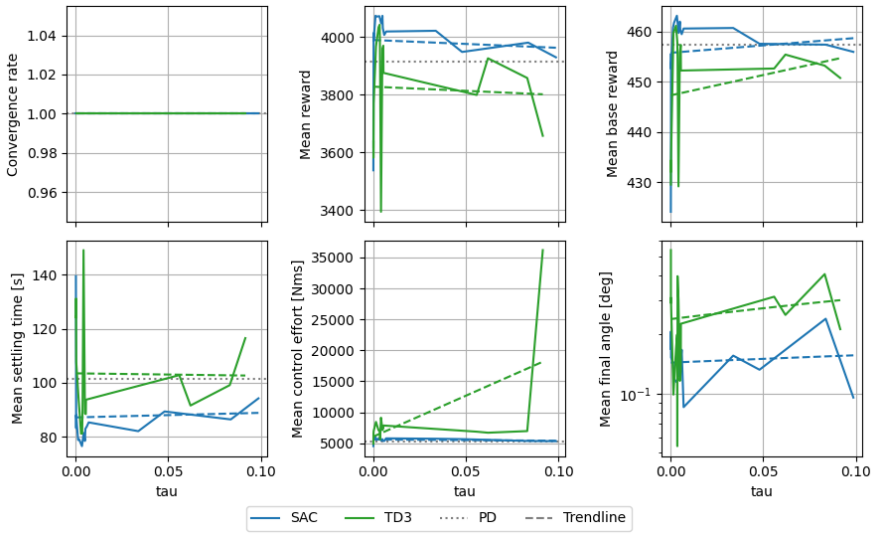


Figure 4.25: Performance of the best agents during their training process plotted against the τ hyperparameter value for SAC and TD3 in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.34$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

In figure 4.25, the effect of τ is shown. It is also shown together with the effect on TD3, as this is a shared hyperparameters. For both agents, the highest peaks are visible at around $\tau = 0.003$ or less (default value of 0.005). However, the sampling size and interval is too low to determine whether this is the result of randomness or whether there is actually an optimum here. In general, for values of $\tau < 0.01$, the performance seems best. Furthermore, on SAC the effect of τ below this value seems to not have much of an effect, while for TD3 its previously discussed variance seems to still be an issue, hence no firm conclusions are possible.

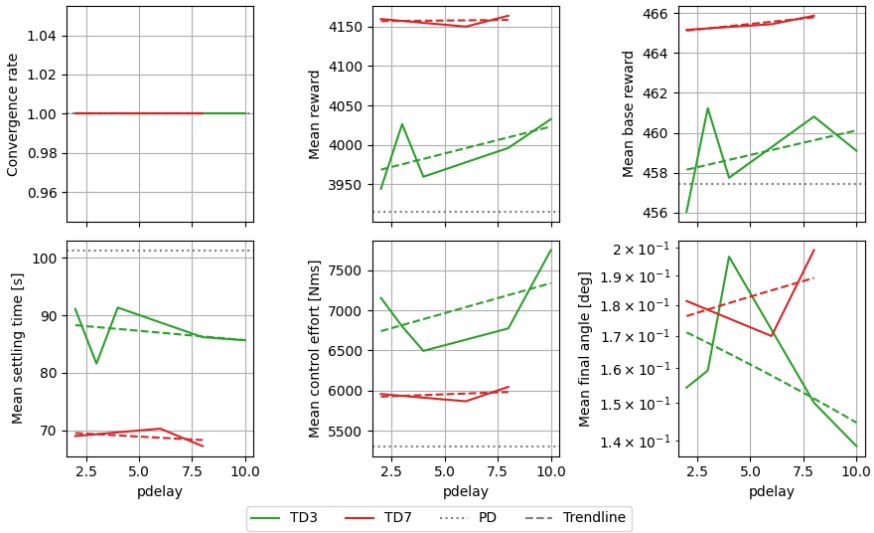


Figure 4.26: Performance of the best agents during their training process plotted against the policy delay hyperparameter value for TD3 and TD7 in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best two fits are $R^2 = 0.91$ and $R^2 = 0.36$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

For TD3, the next hyperparameters are all shared with TD7. The first of these is that of policy delay (default value 2 for both), as shown in figure 4.26. Here, it can be seen that TD7 is not affected much by policy delay, while TD3 seems to benefit from higher policy delays.

The next hyperparameters are about the target smoothing noise. There are two hyperparameters that define its implementation (as a Gaussian around 0), the standard deviation and the clipping value for the noise. By default, this is 0.2 and 0.5 respectively for these values, for both TD3 and TD7. In figure 4.27, the example is shown for the standard deviation, but the insight drawn from it holds also for the clipping (which can be found in appendix A). In TD7, there is no apparent relationship with the hyperparameter and the agent's performance. In TD3, there might be a relationship, but the variance is too high to make any meaningful conclusions.

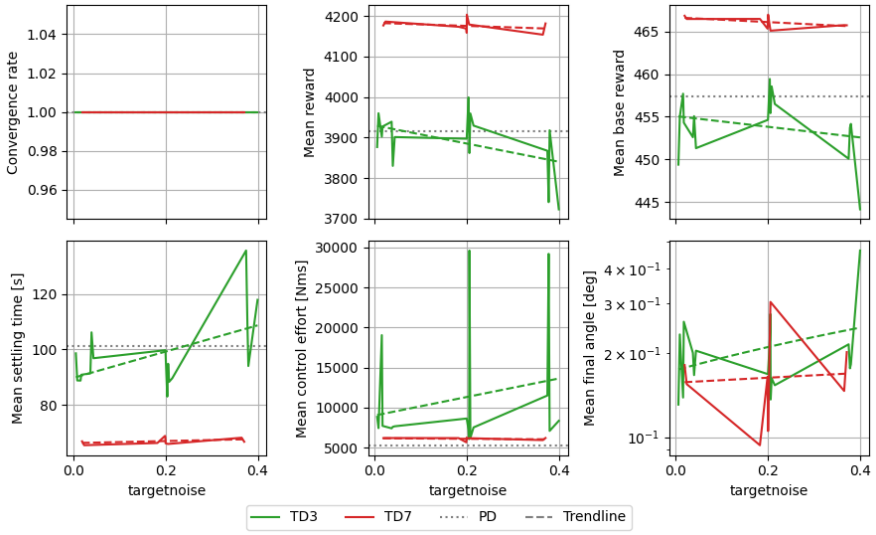


Figure 4.27: Performance of the best agents during their training process plotted against the target smoothing noise standard deviation hyperparameter value for TD3 and TD7 in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.28$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

Finally, TD7 has a few dedicated hyperparameters. The first of these are the criteria reset weight, which is relevant for the checkpointing functionality. The default is 0.9. When varying between 0.8 and 1.0, no significant effect seems to be present. The same seems to be true for the late assessment episode count (default 20, varying between 1 and 25). Their figure can be found in appendix A.

TD7 defines specific hyperparameters for the learned embeddings layer size. From the author paper, the diagram for the critic and actor nets is displayed in figure 4.28, showing the integration of the state-action learned embeddings.

These parameters specify the output size of the encoders $f_t(s)$ and $g_t(z_t^s, a)$ with parameter hdim$_{en}$, and the size of the Linear(s, a) and Linear(s) with parameters hdim$_{cr}$ and hdim$_{ac}$, respectively. The defaults are 256 nodes for each parameter. The result of varying these parameters are shown in figure 4.29.

Interestingly, the same pattern emerges as for the hidden layer node counts in figure 4.24 (where these refer to the hidden layers of the $Q_{t+1}(z_t^{sa}, z_t^s, \phi^{sa})$ and $\pi_{t+1}(z_t^s, \phi^s)$ neural nets as depicted in figure 4.28). A node count of roughly 250 consistently delivers the highest results . However, a lower node count is beneficial for the mean total control effort. This is further evidence that a relationship between these layer sizes and the performance of the agents exists. When looking at the effect of the learned embeddings dimension hdim$_{em}$ specifically, it seems like a lower node size (when hdim$_{ac}$ and hdim$_{cr}$ are kept roughly constant) is especially limiting for performance, for example when looking at the mean reward, mean base reward, and mean settling time. Fujimoto hypothesized that the state-action embedding of TD7 (z^{sa}) has a tendency to expand the action input and increases the likelihood that the value function over-extrapolates on unknown actions. They stabilize this by introducing a clipping on

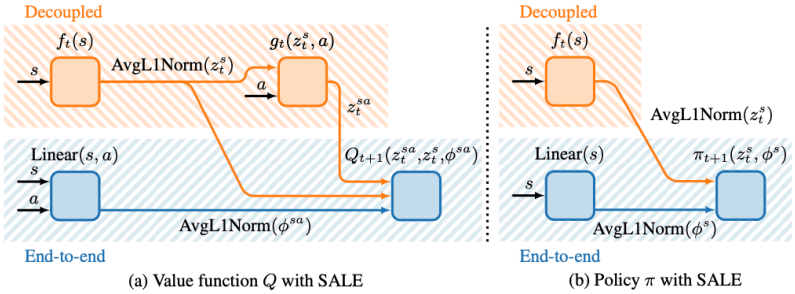


Figure 4.28: Diagram of State-Action Learned Embeddings (SALE), from Fujimoto's original paper [99].

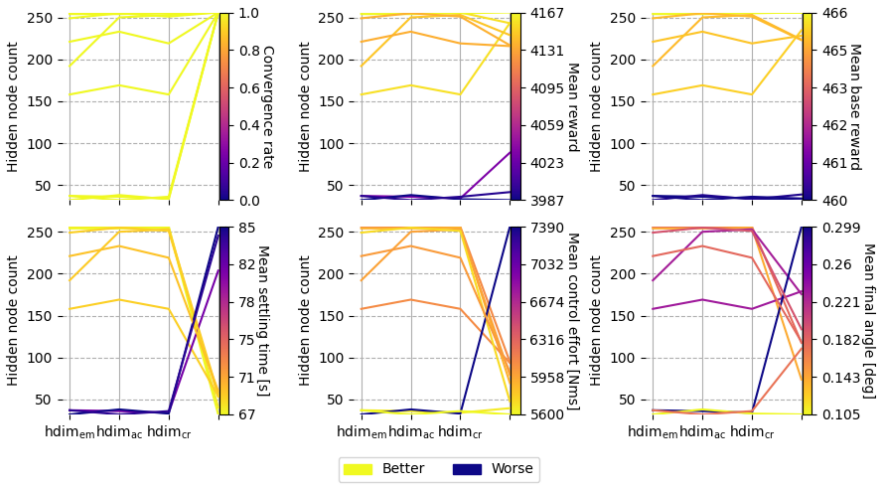


Figure 4.29: Agent performance metrics for different combinations of embedding node count hyperparameter (hdim) values for TD7, for the rigid environment.

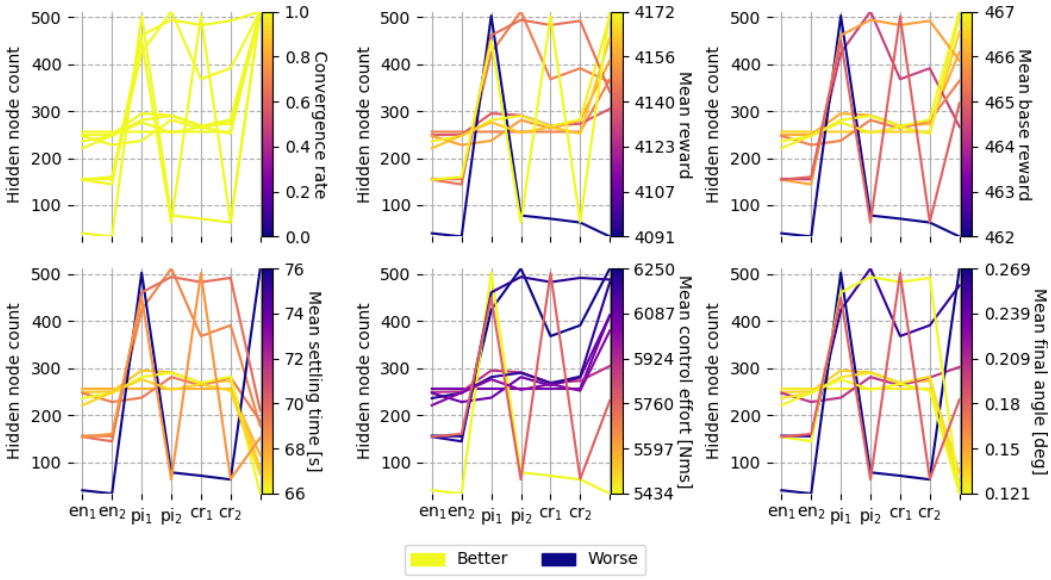


Figure 4.30: Agent performance metrics for different combinations of node count hyperparameter values for TD7 (including encoder layers), for the rigid environment.

the value function target during training [99]. However, the described effect is still visible in the results of this research.

When repeating the experiments for hidden layer size when also varying the layer size of the encoder, the results shown in figure 4.30 tell the same story. The optimum seems to lie in intermediate values of hidden layer dimension (≈ 250).

TD7 incorporates a loss-adjusted prioritized replay buffer [100]. This buffer is characterized by the hyperparameter α, which controls the prioritization. Its default is 0.4. The results of varying this parameter on the performance is shown in figure 4.31. The results seem to indicate that values of α higher than 0.4 have a negative effect on the performance. However, the effect is small, with only a decrease of 1% in mean total reward visible for the results around $\alpha = 0.8$. Furthermore, for $\alpha < 0.4$, the results don't seem to change. Hence, while this hyperparameter might be of influence on the performance, it is not significant and 0.4 seems to be (close to) the optimum.

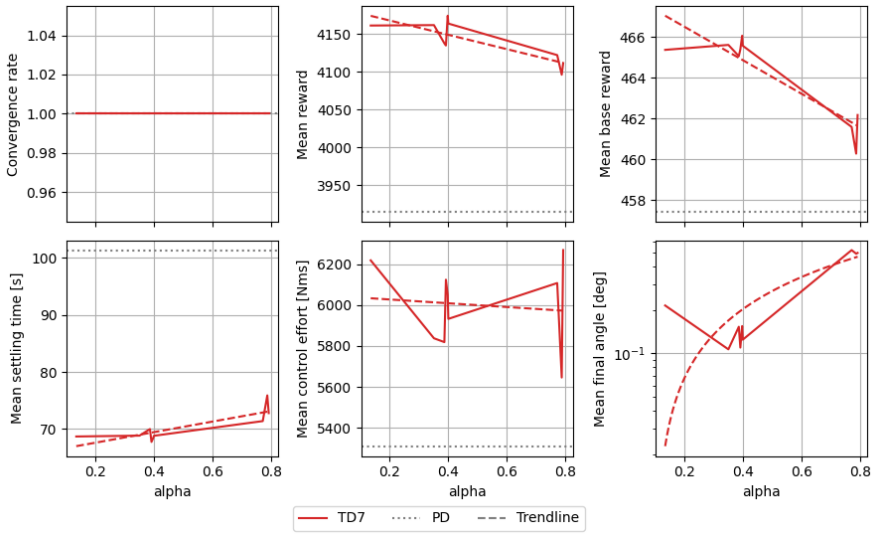


Figure 4.31: Performance of the best agents during their training process plotted against the α hyperparameter value for TD7 in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.80$, which is a reasonable fit.

TD7's final hyperparameter is the target update frequency. This parameter controls the update rule for the target nets (analogous to how Polyak averaging controlled by τ is used for SAC and TD3). From figure 4.32, it can be seen that the relationship is positive: when the update frequency rises, the performance goes up (if the associated increase in mean total control effort is considered to be acceptable). Beyond 250 (the default), however, no further decrease in mean settling time can be observed. A possible explanation is that a larger value (increasing the time between updates) increases the variance in the samples seen by the agent, which in the relatively low dimensional state-space can be beneficial to exploration, speeding up learning. The plateau in settling time could be the minimum average settling time of (near) optimal agents, which cannot be improved further. However, these explanations for the relationship need to be further investigated before they can be accepted.

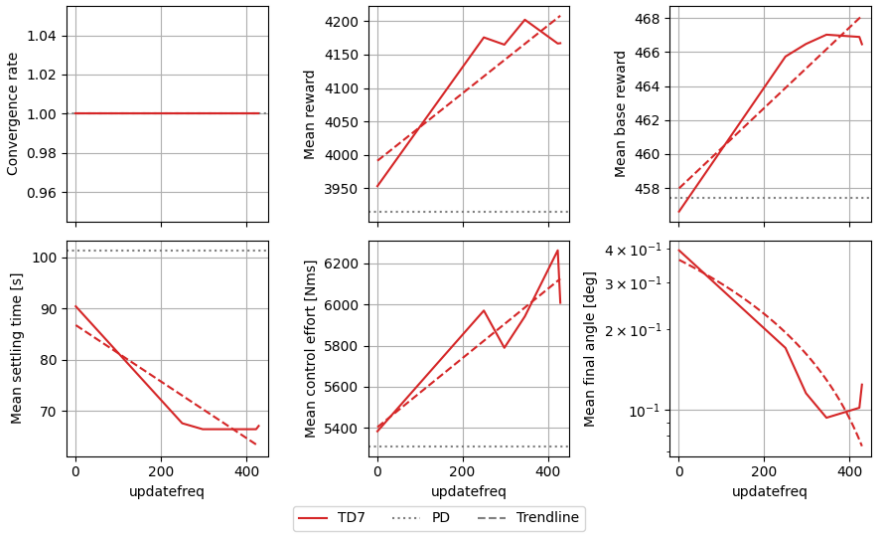


Figure 4.32: Performance of the best agents during their training process plotted against the update frequency hyperparameter value for TD7 in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.88$, which is a strong fit.

4.5. Flexibility effects

The baseline agent performance for the flexible model has been shown in table 4.7. The perturbations are still indexed by table 4.3, which is repeated below for convenience. In table 4.7, it can be seen that even the PD controller (this time the one tuned for the flexible model, as described in table 2.3) does not result in a 100% convergence rate for all the non-gyroscope perturbations (indices 1-3 and 7-10). What also stands out is that the RL agents do no longer achieve a convergence rate of 100% relatively consistently for all the non-gyroscope perturbations, for both the best and final agents.

Table 4.3: Keys for the perturbations listed in table 4.7 (repeated from page 58).

Perturbation	Perturbation index
Unperturbed	1
Inertia scaling	2
Inertia rotation	3
Gyro noise	4
Gyro constant bias	5
Gyro drift	6
Torque misalignment	7
Torque scaling	8
Torque noise	9
Disturbance torque	10

In contrast to the rigid case, PPO fails to deal with the dynamic complexities of the flexible spacecraft. The convergence rates are so low that no agent at any point during training achieved a 100% convergence rate during any evaluation, resulting in a lack of 'best' agents in table 4.7. It might be that if the policy collapse observed in the rigid case can be fixed, that PPO could offer better performance, so it is not necessarily inherent to the on-policy nature of this algorithm.

Another point of contrast is the mean control effort. The PD controller achieves its performance using on average much less total control effort than the RL agents. However, whereas the rigid PD and most agents were able to achieve settling times on the order of 100 s or less on average, it can be seen that the flexible PD only achieves a settling time of 175 s on average. The RL agents trained on the flexible environment reduce this back down to the order of 100 s. This is an indication that there is potential for the RL agents to significantly outperform the PD controller, at least in some metrics.

Table 4.7: Flexible robustness performance of the baseline agents. Shown are mean performance values of the best performing agents during training (best) or the agents at the end of training (final), before filtering for convergence. Note that PPO converged zero agents that converged during any point during the training, so no performant 'best' agent(s) exist for that algorithm, which is why it is omitted. Highlighted in blue are performance values that are equal to or better than the PD controller (right-most column). Note, for the convergence rate and mean episode reward, this means that the values are higher than the PD, but for mean settling time, control effort, and final angle, the values are lower than the PD.

		Best agents			Final agents				
	Index	SAC	TD3	TD7	PPO	SAC	TD3	TD7	PD
Convergence rate	1	1.00	1.00	1.00	0.16	1.00	1.00	0.92	1.00
	2	0.98	0.71	1.00	0.16	0.94	0.65	0.76	1.00
	3	0.46	0.02	0.79	0.18	0.42	0.43	0.62	1.00
	4	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0
	7	0.99	0.97	0.95	0.19	1.00	0.90	0.81	1.00
	8	1.00	1.00	1.00	0.18	1.00	1.00	0.86	1.00
	9	0.58	0.56	0.50	0.14	0.55	0.59	0.60	0.90
	10	1.00	1.00	1.00	0.18	1.00	1.00	0.90	1.00
Mean episode reward	1	4050.8	3950.6	4111	416.9	4040.19	3873.1	4120.3	3362.8
	2	4048.6	3904.3	4106.6	414.2	4036.58	3824.2	4113.7	3361
	3	3666.9	2984.8	4077.4	413.9	3231.6	3761.6	3692.7	3369.7
	4	-390.5	-458.1	-476.4	-220.1	-399.3	-478	-526.8	-332.6
	5	54.1	-25.1	78.4	5.3	78.7	-53.8	96.4	-53.1
	6	-580.0	-527.1	-600.6	-626.7	-572.2	-552.6	-583.5	-570.3
	7	4007.2	3893.7	4058.5	392.9	3997.2	3843.2	4064.5	3268.8
	8	4051.3	3951.3	4109.8	412.5	4038.8	3871.8	4118	3365.8
	9	4049.7	3940	4099.9	410.4	4036.2	3861.7	4101.8	3335.8
	10	4051.2	3951.1	4111.2	419	4040.3	3873.9	4120.2	3360.2
Mean settling time	1	77.7	100.5	104.7	495.1	76.4	107.7	226.7	175.1
	2	119.6	236.6	73.7	495.1	209.0	270.6	319.9	179.9
	3	331.9	499	232	493.7	332.4	347.7	362.4	172.9
	4	499	499	499	499	499	499	499	499
	5	499	499	499	499	499	499	499	499
	6	441.6	393.9	460.2	498.5	434.4	413.1	443.3	430.6
	7	308.5	310.2	405.6	491.6	310.1	384.2	428.2	199.5
	8	84.8	109	103.1	495.1	88.3	113.6	253.5	173.6
	9	496.6	497	497.3	498.1	496.8	497.3	495.6	470.6
	10	79.8	99.6	106.8	495.1	76.4	110	226.4	175.1
Mean control effort	1	13747	14041	12768	19096	13652	16902	14227	4847
	2	14017	23217	12812	19115	14341	26298	14823	4813
	3	36499	62236	19828	18824	45795	41673	27673	4846
	4	89491	95378	93224	58241	89193	95640	93635	88644
	5	13674	24584	13440	19632	13711	32207	13970	6047
	6	85572	78380	90531	91943	84267	82175	87746	84334
	7	13631	14554	14124	18501	13857	16746	14968	5815
	8	13734	14056	12766	18984	13625	16918	14294	4843
	9	14519	16115	14104	19126	14768	19037	15746	5793
	10	13746	14060	12773	19096	13651	16907	14270	4851
Mean final angle	1	0.3	0.3	0.3	10.3	0.3	0.3	0.3	0.3
	2	0.3	0.3	0.3	10.3	0.3	0.3	0.3	0.3
	3	0.4	0.7	0.4	10.3	0.5	0.3	0.5	0.3
	4	44.7	66.7	67.4	21.1	45.5	94.6	119.4	18.7
	5	16.6	21.6	14.2	15.5	14.3	18.7	13.2	37.3
	6	186.3	179.7	174	161.3	177.5	179.9	182.3	159.7
	7	0.3	0.3	0.4	10.3	0.3	0.3	0.4	0.3
	8	0.3	0.3	0.3	10.3	0.3	0.3	0.4	0.3
	9	0.3	0.3	0.4	10.3	0.3	0.3	0.4	0.3
	10	0.3	0.3	0.3	10.3	0.3	0.3	0.3	0.3

Finally, an important note is that while the convergence rate for the RL agents is no longer 100% for the inertia scaling, inertia rotation, torque misalignment, and torque noise perturbations (indices 2, 3, 7, and 9), the mean performance, at least in terms of reward and mean final angle, is still acceptable. The higher settling time could be an artifact of the way the mean settling time is calculated when the convergence rate is not 100%, as discussed in section 2.4.

The first important aspect of the agent behavior to investigate is the behavior for a single episode, instead of only average performance. For this, the best performing baseline agents were simulated for an episode with initial attitude quaternion $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, zero initial rotation, and modal coordinates and rates initialized to zero. For PPO, no acceptable agent was available, so a random final agent was taken. The results are shown for the flexible PD controller, PPO, SAC, TD3, and TD7 respectively in figure 4.33, figure 4.34, figure 4.35, figure 4.36, and figure 4.37.

When comparing the behavior of these different agents, a few things stand out. The first is the behavior shown by TD3 before roughly 105s: the control torque becomes very erratic, is almost always fully saturated, and switches quickly between the two extremes (as shown in the control moment over time subfigure of figure 4.36). This behavior is also observed for different TD3 agents with different seeds and different initial episode states. TD3 is the only algorithm where this behavior shows up. It might explain some of the large variance observed in the learning results later on. Possibly, with the inclusion of regularizing strategies, TD3 can be taught to lose this behavior, but this falls outside the scope of this research.

It can also be observed that PPO is best of all agents in damping the modal coordinate excitations. This is visible in the amplitude of the oscillation of η_2 in the four figures (top right sub-figure). Even though the maximum amplitude is not the same for each agent (roughly 210, 220, 60, and 200 for PPO, SAC, TD3, and TD7 respectively) initially, it can be seen that PPO dampens this amplitude significantly to a value of roughly 140 (reduction of one third) by the end of the episode, while the amplitude for TD7 is still larger than 160 by the end of the episode (reduction of one fourth). Hence, if the performance of PPO is made higher and more stable somehow, then PPO is a promising option due to the damping behavior it displays.

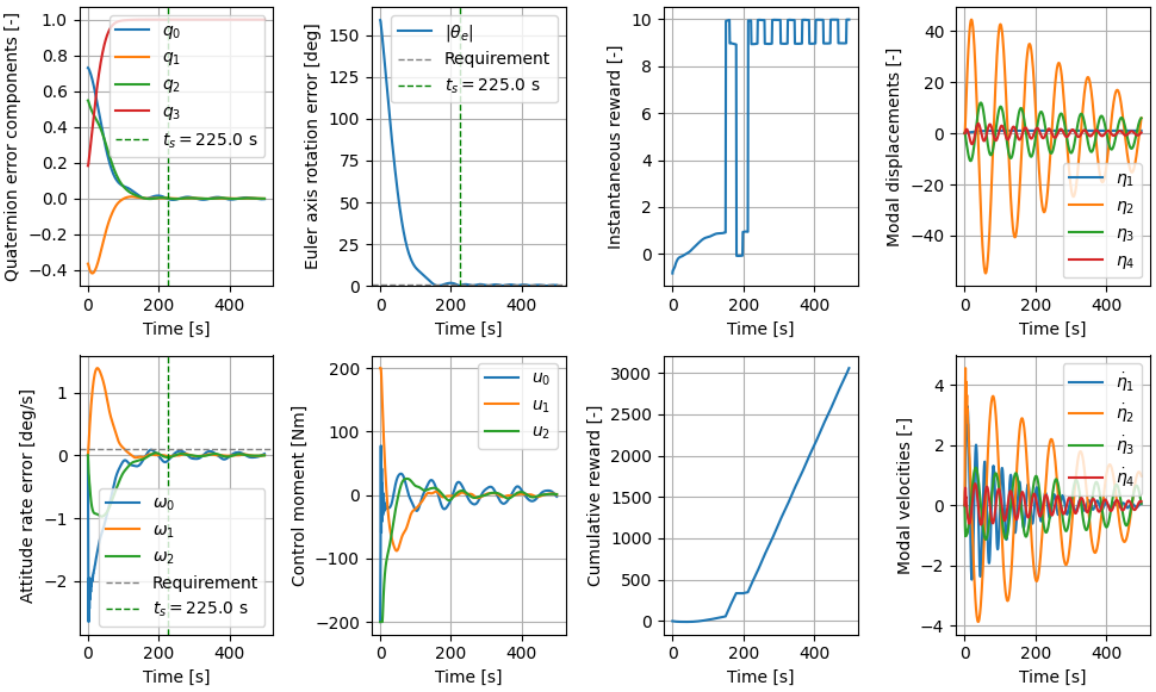


Figure 4.33: A single episode for the flexible model, controlled using the PD controller. Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

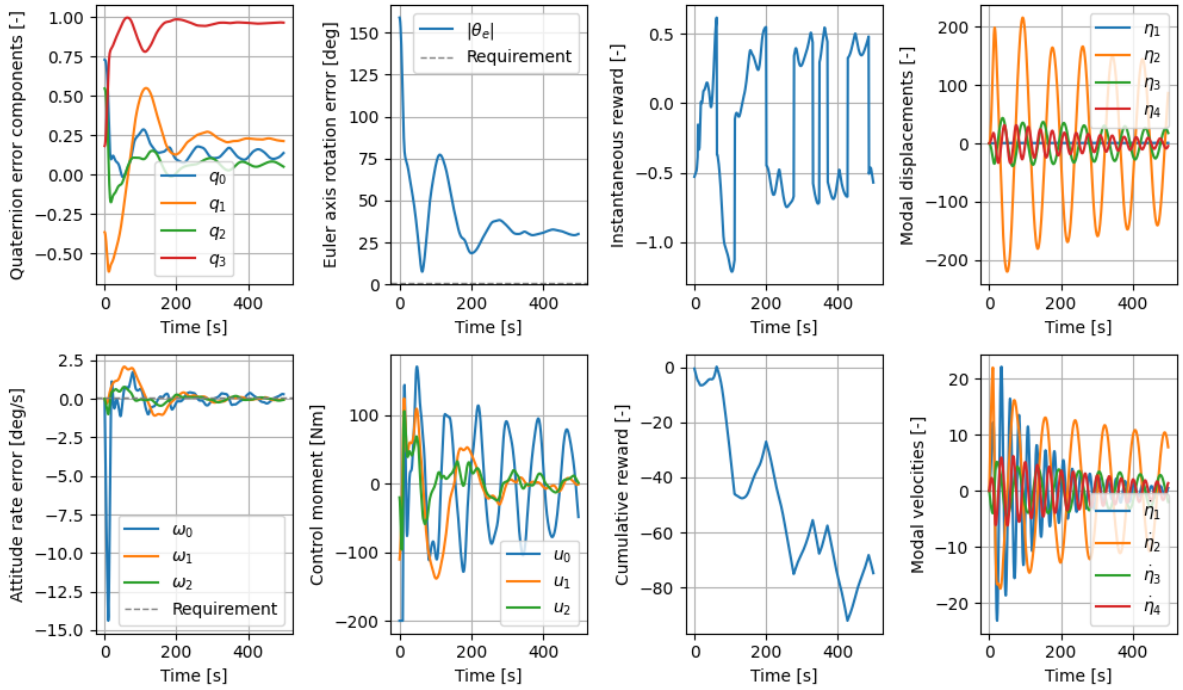


Figure 4.34: A single episode for the flexible model, controlled using a final PPO agent.Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

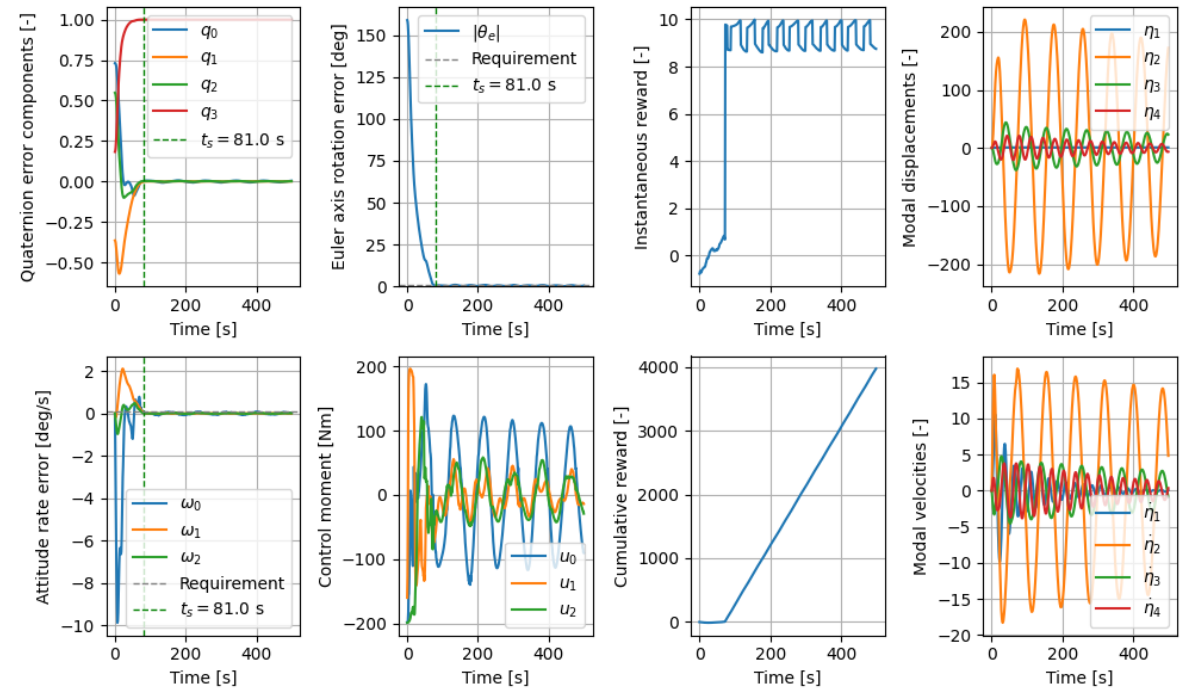


Figure 4.35: A single episode for the flexible model, controlled using the best SAC agent. Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

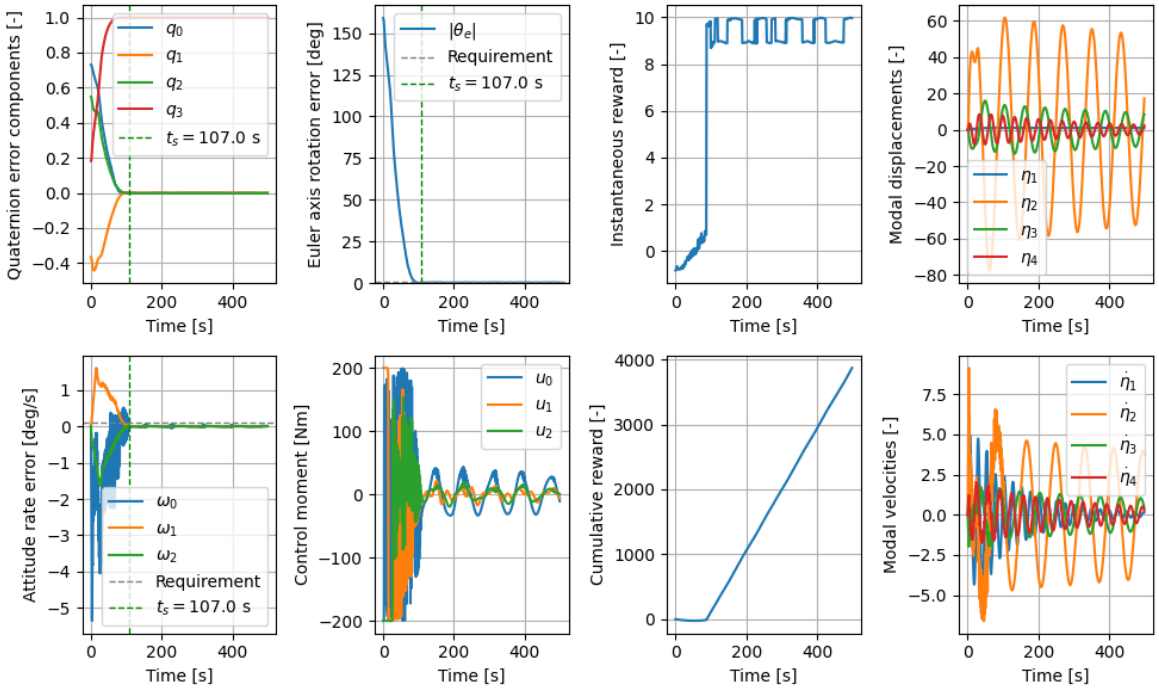


Figure 4.36: A single episode for the flexible model, controlled using the best TD3 agent. Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

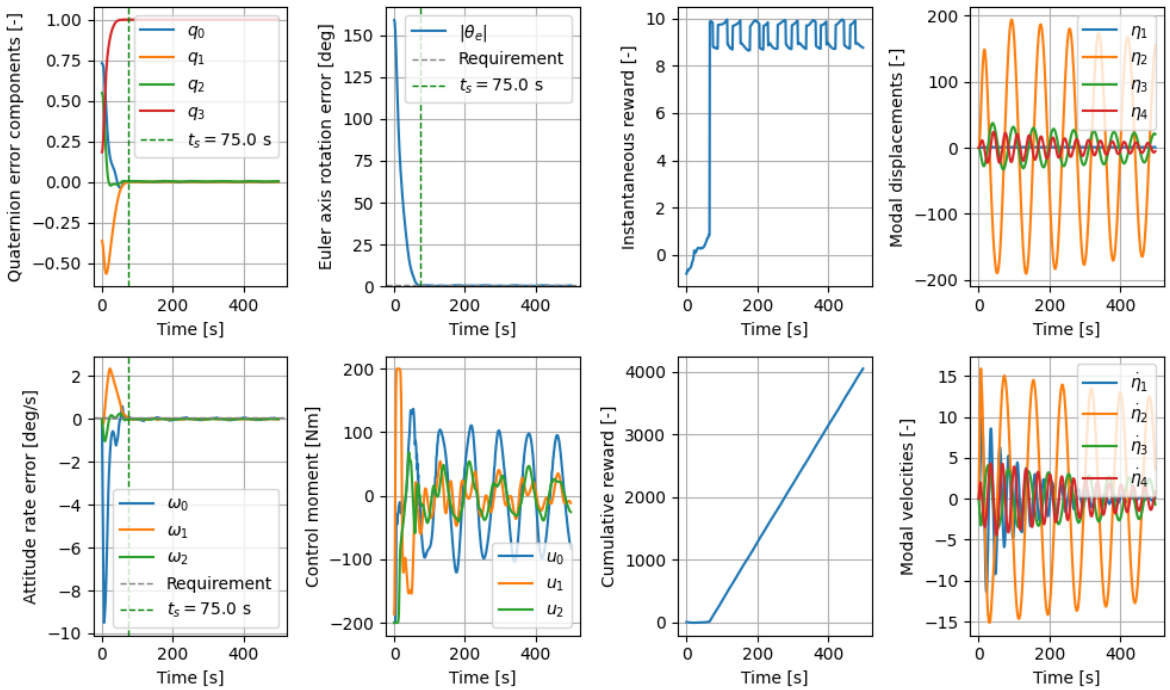


Figure 4.37: A single episode for the flexible model, controlled using the best TD7 agent. Initial quaternion is $q_0 = \begin{bmatrix} 0.73029674 & -0.36514837 & 0.54772256 & 0.18257419 \end{bmatrix}^T$, with other initial values all zero.

When looking at PPO in more detail, it excites the modal vibrations the most, together with SAC and TD7. At the same time, while the other two display performance within the requirements, PPO is not able to deal with this effectively, resulting in a (roughly) steady state error of larger than 25°.

A key observation of all the RL agents is that an oscillation in the control torque remains until the end of the episode. This oscillation is largest in u_0, and matches the frequency of η_2 phase-shifted by half a period. This indicates that the control torque is directly compensating for the oscillation, but is not effectively damping it. The continued oscillation of the control torque also shows up for the PD controller, but is of much lower amplitude. This might explain the large difference in total control effort seen in table 4.7 between the PD controller and the RL agents.

The off policy algorithms are all not damping the modal vibrations effectively. However, these agents are able to point the spacecraft to the target within the requirements and with a low settling time, and keep it pointed there. The agents still compensate the oscillation (TD7 and SAC more than TD3), but as long as total control effort is not important and the structures of the chaser, target, and capture mechanism can withstand continuous oscillations, this could be acceptable.

A final interesting note is about the internal vibrations and the influence of the 1 Hz control sampling. From equation (2.18), when converted to Hz, the eigenfrequencies of the four different modes are set to $f_\eta = \begin{bmatrix} 0.01222469 & 0.01756752 & 0.0298145 & 0.04057814 \end{bmatrix}^T$ Hz. After the initial slew maneuver (and the resulting excitation of the vibrational modes), the agents are not further exciting the vibrations, so no resonance takes place. This does not generalize directly to other (especially higher) eigenfrequences and modes, however, and should be checked before an RL agent can be used in the attitude control of a spacecraft.

Next, the learning process was assessed for the flexible model. The baseline case (with minima and maxima shading) has been displayed unfiltered in figure 4.38. For this flexible case, similar observations can be done as for the rigid case: SAC and TD7 have the highest performance (with TD7 having the highest performance), SAC is most consistent, and PPO never achieves a convergence rate of 1.0.

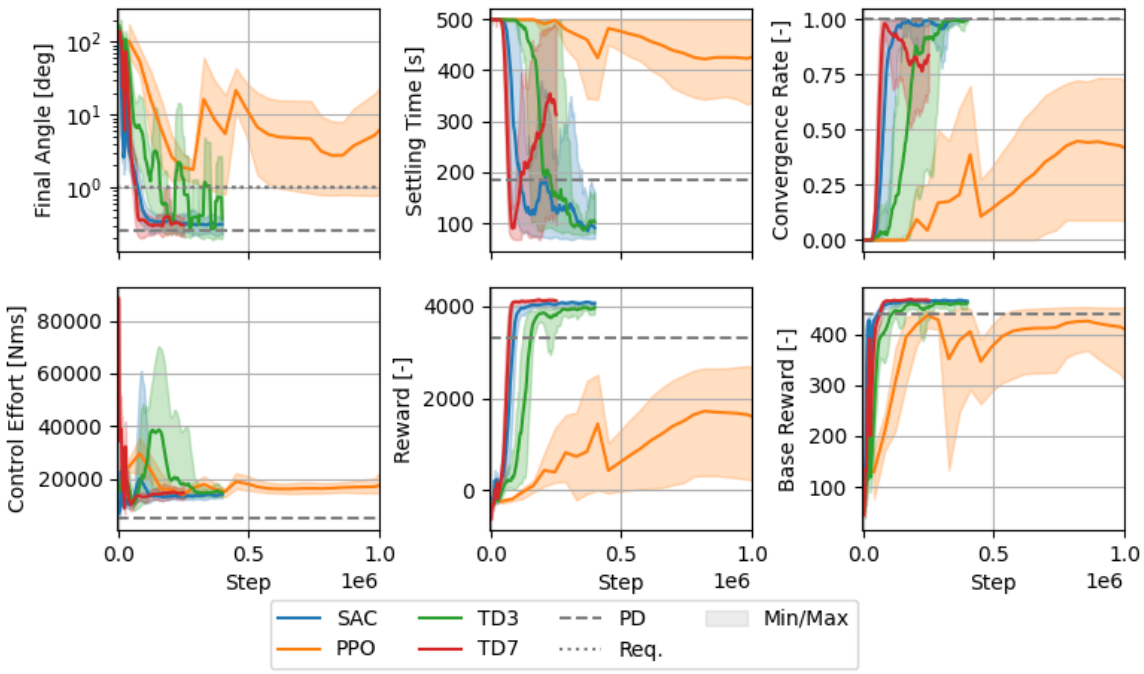


Figure 4.38: Average, minimum, and maximum results for multiple learning processes with the four different algorithms in the flexible case, with 10-step window smoothing applied.

When looking at the computational time of the flexible model, as shown in table 4.1 and table 4.9, the computational time is slightly higher than the rigid case. This is explained by the increase in model complexity and hence computational complexity in the environment. However, the difference in reward at 100k steps becomes larger, as TD7 and SAC are the only agents that shows performance corresponding to a convergent agent. TD3 learns more slowly, as is also visible in figure 4.38, and PPO is having trouble properly learning at all. TD7 is also still faster than TD3 in terms of PD exceed time. Hence, in terms of computational efficiency, TD7 is always (both in the rigid and flexible cases) better than TD3. SAC still is the most computationally efficient algorithm.

Table 4.8: Computational time metrics for the flexible baseline case.

Agent	Steps per second	Training time [s] (total steps)	Time per 100k steps [s]
PPO[1]	315.8	11206 $\pm$ 1154 (3.5M)	316.7
SAC	48.2	8309 $\pm$ 293 (400k)	2075
TD3	45.8	8873 $\pm$ 1142 (400k)	2183
TD7	18.5	13558 $\pm$ 876 (250k)	5405

Table 4.9: Comparative computational time metrics for the flexible baseline case.

Agent	PD exceed steps	PD exceed time [s]	Reward at 100k steps
PPO	-	-	-186.6 $\pm$ 20.0
SAC	65500 $\pm$ 9682	1359	3942.1 $\pm$ 64.2
TD3	125439 $\pm$ 14814	2739	1230.8 $\pm$ 783.1
TD7	49121 $\pm$ 4680	2655	4110.2 $\pm$ 33.9

The above results can be summarized as follows. PPO does not perform sufficiently for the flexible model at all. TD7 is more computationally efficient than TD3 and achieves better results than TD3. It is also the best performing model. However, if performance is not critical, or if computational efficiency is critical, SAC might be a better choice, as it is as robust as TD7 and requires a much lower control effort.

The requirement coefficient of the reward function c_r does not affect the performance differently in the flexible case when compared to the rigid case. The sensitivity graphs for this are hence shown in appendix A. The same is true for c_a and c_p, no significant difference with the rigid case is observed and hence those results can also be found in the appendix. The data for c_a and c_p is also still of too high variance to draw any reliable conclusions, just like in the rigid case.

One exception to this exists. Interestingly, SAC's performance in terms of settling time for the unperturbed case becomes much worse when $c_p = 0$, as shown in figure 4.39. The standard deviation, while still large, is not larger than the change itself, in contrast to the other results for c_p.

This result seems to indicate that c_p can be beneficial (or in case of SAC, even required) for agent convergence. This is in contrast to the rigid case. This extra penalty for a non-improving state change might be beneficial for generalizing a policy for the more complex flexible model.

In terms of domain randomizations, some algorithms had trouble to converge with the domain randomizations applied. For example for the inertia rotation randomization, only 1 TD7 agent did, and no other algorithms converged. This means that TD7 has too few converged agents to make a statistically significant comparison between mean performance, robustness, and the application of this domain randomization. For inertia scaling, the standard deviations are larger than the changes of the means. Hence, no straightforward conclusion can be drawn from the data. Both of these are hence included in appendix A.

For the gyroscope perturbation domain randomizations, again, none of the algorithms result in performant policies. The sensitivity analysis is nonsensical for unconverged agents. Hence, these figures

[1]The metrics for PPO are only for unfiltered agents, as filtering the agents in case of PPO would result in all the data being removed, as no single agent achieved 100% convergence at any evaluation point during training.

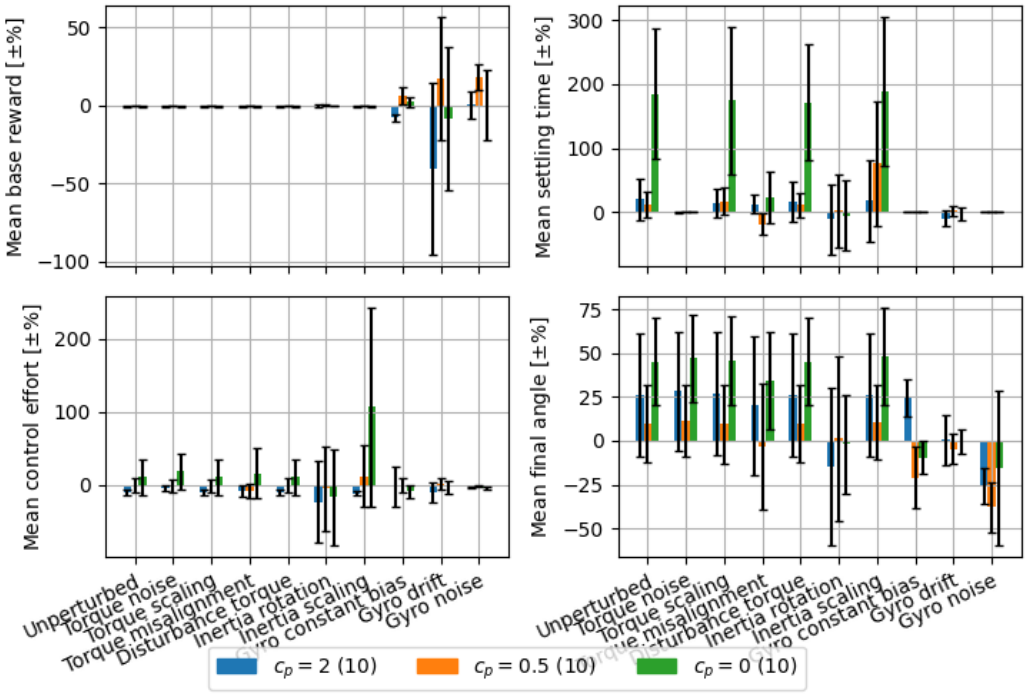


Figure 4.39: Performance of SAC agents trained with variations in c_p in the reward function. Shown are the changes with respect to the baseline under all the investigated perturbations, only for the flexible environment. Bars indicate one standard deviation.

are omitted. These domain randomizations are hence not suitable in both the rigid and flexible cases to deal with the gyroscope perturbations. Two possible insights can be drawn from this. Either the perturbations tested are too large, and the domain randomizations might still be effective for smaller magnitude perturbations, or the agents are simply not capable of dealing with these perturbations without further modifications or additions to the control loop.

The attitude representation variations for the flexible case do not improve performance. Just like the rigid case, in terms of reward, if there is any relation, it is a net negative for both the Euler and MRP representations when compared to the quaternion representation. However, in the flexible case, the standard deviation of the data is even larger. Hence, no clear insights can be taken from the data. The results are shown in appendix A.

As visible in figure 4.40, the misalignment domain randomization does not improve the robustness performance to the torque misalignment with sufficient certainty. Interestingly, the settling time for the unperturbed case does decrease, with a low standard deviation (so higher certainty), for TD3 and TD7 (based on 8 and 10 convergent agents, respectively). The reward does not change, and the mean control effort goes up by 4.4 and 6.9% respectively. Hence, this domain randomization might contribute to a policy that is higher performing in the unperturbed case.

This is clear because the domain randomization does help to improve the convergence rate for SAC, TD3, and TD7, as summarized in table 4.10. This shows that the RL agents respond well to the domain randomization, and achieve better generalization when it is implemented. It is further evidence that domain randomization can in some cases help improve performance. Still, the settling time of the PD controller is lower than the settling times of any of the RL agents for this perturbation.

The torque scaling, torque noise, and disturbance torque domain randomizations result in performances with standard deviations larger than the differences in mean between the DR-agents and the baseline agents. Hence, no confident insights can be drawn from these results. The sensitivity analysis figures are included in appendix A.

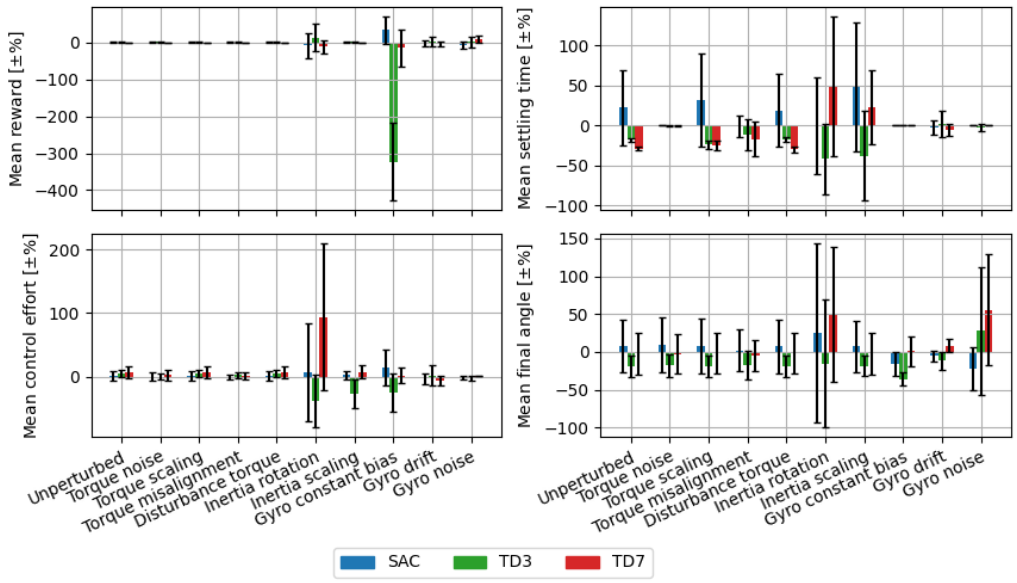


Figure 4.40: Performance of agents with domain randomization applied in the form of torque misalignment, with respect to the baseline performance, in the flexible case.

Table 4.10: Summary of performance for the PD controller and the baseline (BL) and torque misalignment domain randomized (DR) SAC, TD3, and TD7 agents. The performance shown is the performance of the controllers for the flexible case when the torque misalignment perturbation is applied.

	PD	SAC (BL)	SAC (DR)	TD3 (BL)	TD3 (DR)	TD7 (BL)	TD7 (DR)
Convergence rate	1	0.99	1	0.97	1	0.95	1
Episode Reward	3268.8	4007.2	3999.6	3893.7	3908.9	4058.5	4023.9
Episode base reward	437.7	459.2	458.7	453.6	453.7	460.6	458.7
Settling time [s]	199.5	308.5	305.3	310.2	275.7	405.6	337.6
Control effort [Nms]	5815.8	13631	13592.7	14554.2	14878.3	14124.3	14326.5
Final angle [deg]	0.3	0.3	0.3	0.3	0.3	0.4	0.3

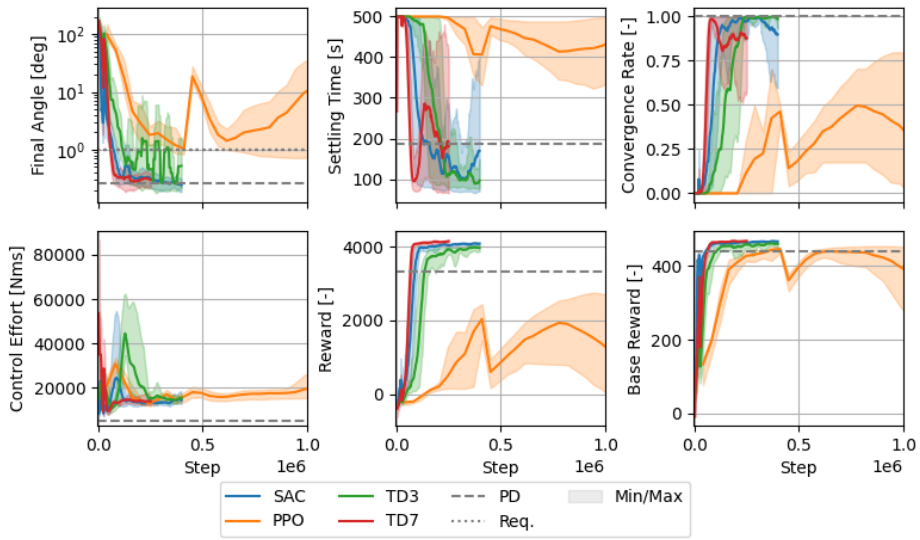


Figure 4.41: Average, minimum, and maximum unfiltered results for multiple learning processes with the four different algorithms in the flexible case with the full observation vector available, with 10-step window smoothing applied.

The only domain randomization that is unique to the flexible case is the full observability domain randomization, as described in equation (2.22). The learning curves for this case are shown in figure 4.41, and the sensitivity analysis for all the perturbations is shown in figure 4.42.

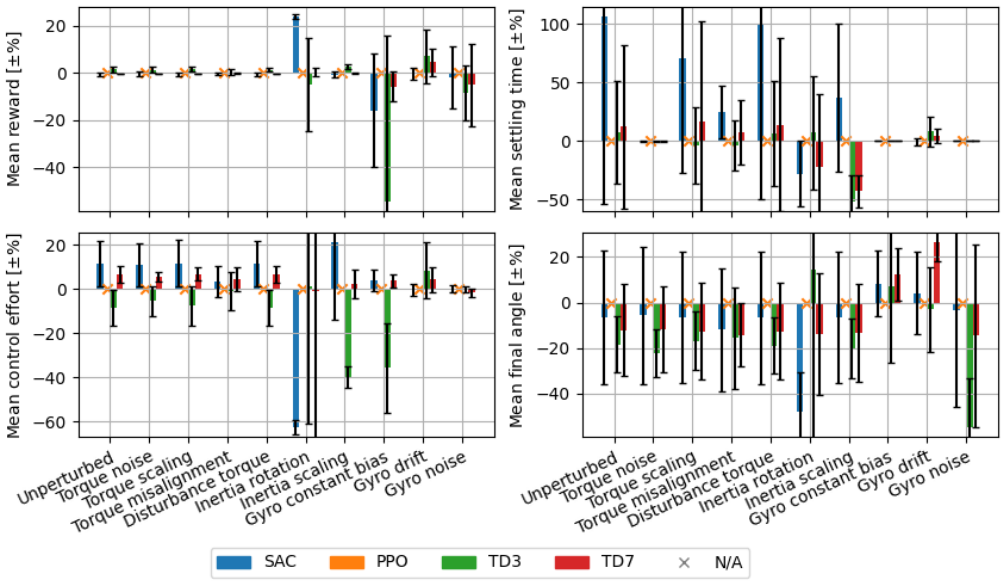


Figure 4.42: Performance of agents with full observability enabled, with respect to the baseline performance, in the flexible case, for the (filtered) final agents.

The expectation would be that with the full observability enabled, the agents would be able to generalize better. The risk is that the agents will overfit to the sampled data more. Both of these cannot be clearly identified or disproven based on the data. The reward is slightly higher for SAC with full observability enabled, but the variance is large. Furthermore, due to the latent state action embeddings of TD7, the effect of the full observability is expected to be lower. The reward does not meaningfully change for TD7. However, again, the variance is large.

For all agents, it is clear that the variance in the fully observable case is relatively high. Drawing any firm conclusions is hence difficult from these results. One remark is that the performance of SAC seems to improve significantly when looking at the inertia rotation perturbation with full observability. This is another indication that this perturbation can be dealt with appropriately in several ways, but that drawing generalized conclusions about what design decisions work well in all cases is still difficult.

The insight for the hyperparameters from the rigid case transfers relatively well as well. For example, for the exploration noise hyperparameter (which is shown in figure 4.43) still exhibits a rise in performance for TD3 with a larger exploration noise. However, the effect disappears for TD7 (for which the sample range is too small to draw meaningful conclusions anyway). Furthermore, the changes of PPO cannot be attributed to solely the change in hyperparameter, as PPO does not perform consistently in the baseline case.

The same is true for other hyperparameters, such as γ shown in figure 4.44. The relationship between γ and the performance seen in the rigid case is preserved for the flexible case for TD3, but not for PPO. SAC's sampling range is too small to make meaningful conclusions, and TD7 did not result in convergent agents for this study.

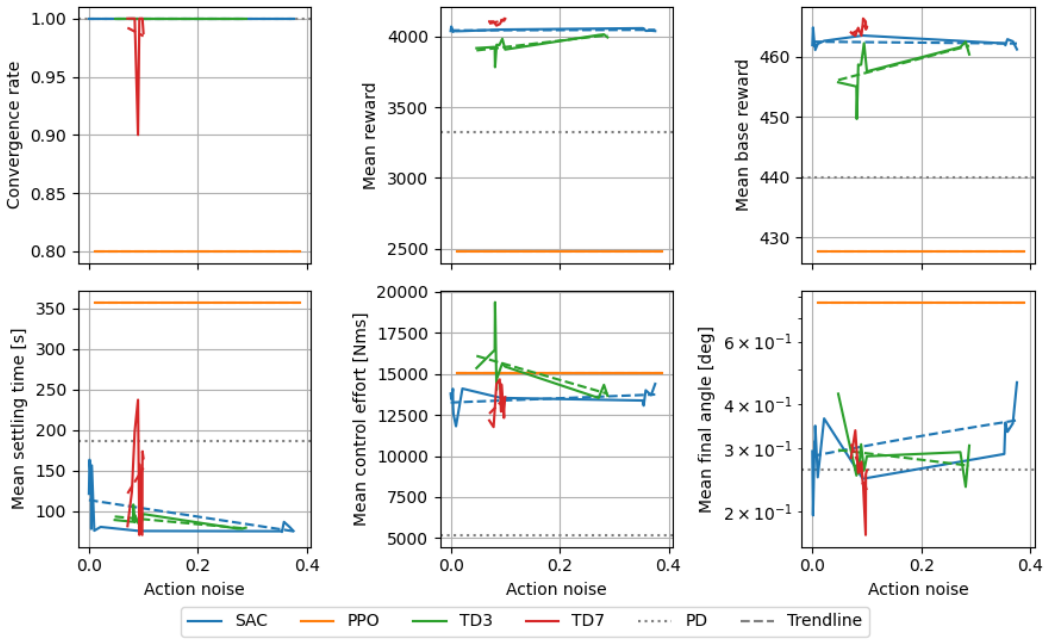


Figure 4.43: Performance of the best agents during their training process plotted against the exploration noise hyperparameter value for all four algorithms in the flexible unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.67$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

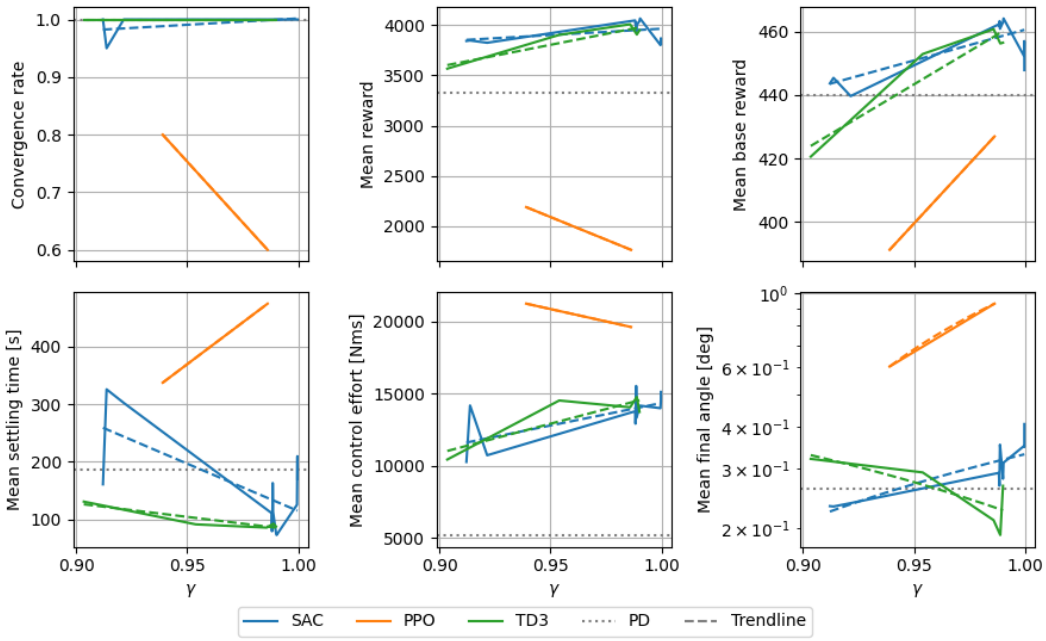


Figure 4.44: Performance of the best agents during their training process plotted against the γ hyperparameter value for all four algorithms in the flexible unperturbed environment. The trendline for TD3 in the mean episode reward figure has an $R^2 = 0.87$, indicating that the relation is (roughly) linear. However, the trendline of SAC has $R^2 = 0.18$, which is a very bad fit. The trendlines have still been included, to provide a sense of the first-order trend.

The learning rates still do not exhibit a clear straightforward relationship to performance. Based on the results in figure 4.45 (here, the final agents have been plotted instead of the best agents, as the algorithms did not result in many acceptable agents), the optimum learning rates are slightly lower than the rigid case, especially for SAC and TD3. This might be explained by the higher model complexity of

the flexible model, which is more difficult to generalize for.

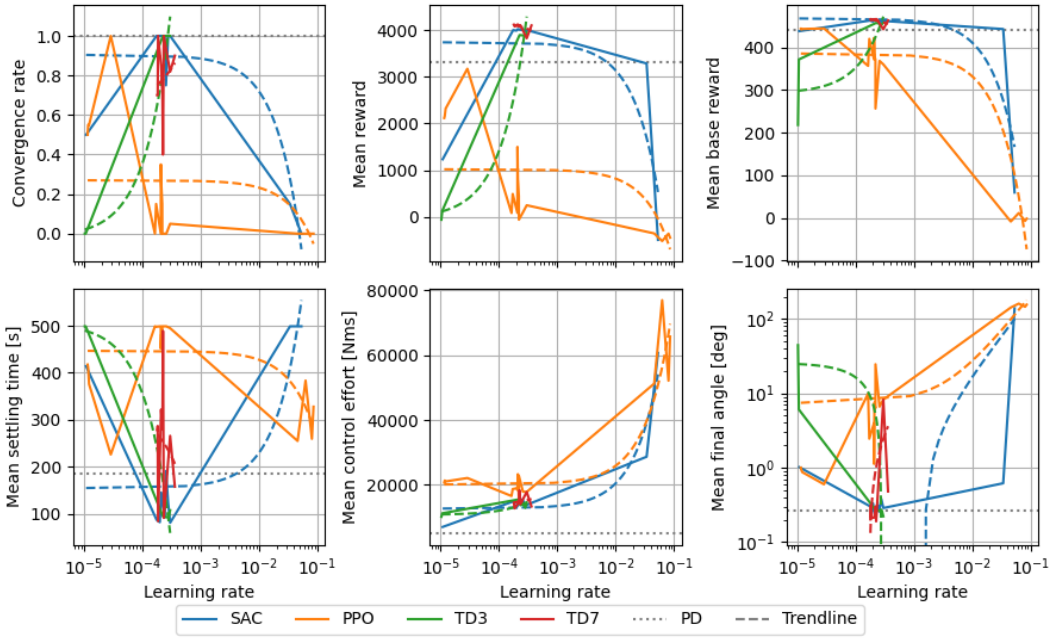


Figure 4.45: Performance of the final agents during their training process plotted against the learning rate hyperparameter value for all four algorithms in the rigid unperturbed environment. Of all the trendlines shown in the mean episode reward figure (all linear trends), the best fit is $R^2 = 0.30$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

The other common hyperparameters (hidden layer sizes and activation function) did not display meaningfully different results than the rigid case. The same is true for the hyperparameters specific to one or multiple of the algorithms. Hence, these figures have all been included in appendix A.

5

Conclusion and recommendation

This chapter takes the insights gathered in the results chapter, and reconnects them with the specific research questions. The implications of the answers to the questions are then also reflected upon. First, in section 5.1, the conclusions are laid out. Then, finally, in section 5.2, the recommendations are formulated and the research is wrapped up.

5.1. Conclusions

This section draws the relevant conclusions for the research. This is done separately for the rigid and flexible cases, because the differences between the spacecraft categories they accurately represent are quite large. Some ADR missions will not be meaningfully flexible at all, and might be sufficiently modelled using the rigid body. In that case, it is much clearer to just see the rigid results. The flexible results and conclusions are an extension to this, for readers to which a higher fidelity model is more relevant.

In the rigid case, the baseline controller performs surprisingly well. All algorithms are able to produce agents that achieve a performance at least on-par with the appropriately tuned PD controller, which in case of this relatively simple model already delivers arguably very good results.

When the robustness is tested, the agent shows robustness to some of the perturbations. Especially a small change (the perturbations were on the order of at most a few percent, as discussed in section 2.5.1) in model parameters are manageable to a more or lesser extent by the agents. This includes perturbations in the inertia tensors (representing uncertainty in the mass distribution) and uncertainties in the torque response (by slight misalignment, random noise, and a disturbance torque).

Where the agents struggle is with perturbations that are applied directly on the input of the agent. This includes all the gyroscope perturbations. The agents are not resistant to these perturbations at all, or at least to the magnitude of the perturbations used in this research. It could be expected that with fine-tuning, the RL agents could learn the bias of the gyroscope, and perform well despite this bias. This would warrant further research, however.

The robustness comparison with the PD controller is also quite positive. The PD controller fails in the cases where the gyroscope data is perturbed as well, as can be seen in table 4.4. Furthermore, in all cases where the PD controller doesn't fail, SAC and TD7 outperform PD in terms of mean episode reward and settling time, when looking at the best agents.

Hence in conclusion, for the rigid case, the baseline agents do not offer a meaningful robustness improvement, as the PD controller is often more robust. However, in the cases where the agents are robust, the demonstrated performance of at least SAC and TD7 is better than the PD controller on its own.

This conclusion is somewhat adjusted when looking at the performance for one of the domain randomized agents. For example, when looking at the performance of the inertia rotation domain randomized

Table 4.5: Summary of performance for the PD controller and the baseline (BL) and inertia rotation domain randomized (DR) TD3 and TD7 agents. The performance shown is the performance of the controllers for the rigid case when the inertia rotation perturbation is applied (repeated from page 64).

	PD	**TD3 BL**	**TD3 DR**	**TD7 BL**	**TD7 DR**
Convergence rate	1	0.82	1	1	1
Episode Reward	3940	3827.8	3658	4148.7	4096.1
Episode base reward	458.4	452.7	448.1	465.3	464.5
Settling time [s]	98.6	169.7	121.1	69.8	71.6
Control effort [Nms]	5073.6	21418.7	13923.6	6341.5	10803.2
Final angle [deg]	0	1	0.5	0.2	0.1

agents, the convergence rate of TD3 is brought back to 1.0 for when evaluated in an inertia rotation perturbed environment (instead of 0.82), and its settling time performance of is also improved significantly. This is summarized in table 4.5, repeated here. TD7 is also included in this table to demonstrate that the performance does not improve in all cases, and can even degrade significantly (as seen by the rise in mean total control effort). This means that domain randomizations should not be applied without careful thought or without trying the performance when omitting the domain randomization.

From the above, it is clear that domain randomization, when set up correctly, could have a robustness improving effect on RL-based controllers. Furthermore, it is also clear that improvements upon the baseline agents are difficult. Finally, compared to the PD controller, the best RL-based controllers achieve a higher reward and significantly lower settling time. However, the PD controllers often require a lower mean total control effort.

An important remark should be highlighted in the summarized conclusions above: The reliability of the RL-based agents is debatable at best. While the convergence rate of the best agents is 100% in the unperturbed case, when perturbations are applied this often drops below 100%. Furthermore, the final agents (at the end of training), don't get 100% performance at all. From a theoretical perspective, a proof of convergence (given even a simple model, for example a linearized one), is difficult if not impossible to obtain. Hence, the research sub-questions are answered for the rigid case as follows:

- *SQ1: How can model-free reinforcement learning algorithms be effectively adapted for the design of a controller for post-capture control in the context of active debris removal missions?*

  → A quaternion state representation is most suitable for the attitude. This is combined with a reward function that is exponentially shaped towards the target, penalizes the control torque by at most half the magnitude of the maximum shaped reward, and gives a bonus reward one order of magnitude larger than the rest of the reward when within the requirements. A bonus for improvement of the state (with respect to the prior state) does not affect the learning performance.

  → All four model-free reinforcement learning algorithm are able to generate policies that are more performant than a hand-tuned PD controller when using conventional hyperparameters, if the post-capture attitude control problem can be modelled as an idealized rigid spacecraft without perturbations.

  → Off-policy algorithms are generally more performant than on-policy algorithms. TD7 is the most performant algorithm. SAC's learning process is most reliably increasing the agent performance, with a reasonable computational cost and the lowest computational cost of the investigated off-policy algorithms. PPO offers the lowest computational cost of all the RL algorithms.

  → The optimization of hyperparameters can in some cases offer either improved performance or a policy that is smoother. However, this depends highly on the models and algorithms used, and is not the case in the general sense.

- *SQ2: To what extent is the stability and convergence of a model-free reinforcement learning-based controller reliable?*

  → The stability and convergence of the agents during the learning process is not strictly increas-

ing. The best performing agents should be saved during training and separately evaluated to achieve the highest convergence rate.

→ No strict guarantees on stability and convergence exists for these agents. However, when testing agents for many randomly initialized episodes on the simple rigid model, all algorithms could produce agents for which the convergence rate hit 100% at least once during training.

- *SQ3: How do inherent uncertainties in the dynamic and kinematic properties of the system affect the performance of a model-free reinforcement learning-based controller in the post-capture phase of an active debris removal mission?*
  → The algorithms are able to deal with some perturbations effectively, like disturbance torques, inertia scaling, inertia rotation, torque scaling, torque misalignment, and torque noise.
  → A hand-tuned PD controller is also able to perform with these perturbations. The agents hence do not offer meaningful robustness improvements over the PD controller in the rigid case. In the cases where both the PD controller and a reinforcement learning agent are robust to the perturbation, the PD controller achieves this with a lower mean total control effort.
  → In some cases, the application of domain randomization can improve the robustness of the reinforcement learning agents to these perturbations.
  → The agents fail to deal with perturbations that directly affect the input of the policy (perturbations in the gyroscope). These uncertainties deteriorate the performance of the agents to a convergence rate of 0%. The robustness is not improved by using domain randomization in this case.

When looking at the flexible case, the conclusions for the rigid case above translate relatively well. In this case, the inter-agent comparison can be summarized by stating that TD7 is the most performant algorithm, followed by SAC. SAC and TD7 are the two most robust algorithms, where sometimes TD7 is more robust and sometimes SAC is more robust. In case a reinforcement learning agent is robust, in contast to the rigid case, the RL agents can offer significantly better performance than the PD controller, for example in the settling time, for some of the perturbations. Finally, PPO in the current form of implementation is entirely inadequate for the flexible model.

An important distinction with the rigid case is that the baseline SAC and TD7 agents now no longer consistently achieve a 100% convergence rate for the inertia scaling and rotation nad torque misalignment and noise perturbations. The PD controller still achieves 100%, except for under the torque noise perturbation.

In terms of computational cost, TD7 is still the most computationally expensive in terms of compute time per step. However, as relatively more computational time is spent on the spacecraft model instead of training the agent, the difference is a little smaller than in the rigid case. Furthermore, just like in the rigid case, TD7 is much more computationally efficient, meaning that its agents exceed the PD performance in a lower wall time than TD3. SAC is still a good balance between good sample efficiency, performance, and overall computational cost.

In the rigid case, the domain randomization only offered meaningful performance improvements in some specific cases. This also generally holds for the flexible case. For example, within the misalignment domain randomization. This randomization showed that the convergence rate can be increased (and hence, the reliability), and that performance metrics can improve (in this case, the settling time).

An important distinction with the rigid case is the insight that the non-improvement penalty reward c_p could have a positive effect on agent performance, which was observed to be the case for SAC.

The full observability variation that was tried does not seem to meaningfully improve performance. The agents, especially TD7, are able to produce agents that perform very well for the reward functions specified even in the default partially observable case (which even though improvements to the algorithms in the POMDP case exist [144] is generally in agreement with previous findings about policy gradient algorithms[145][57]). The fact that full observability is not required for acceptable agent performance, at least under the requirements of this research, is positive as it means that any ADR spacecraft would not require a sensing system to measure the vibrations. This lowers complexity and ultimately the cost of a mission.

Importantly, only PPO learned policies that dampened the oscillations in the modal displacements significantly. The other algorithms simply compensate for the internal vibration with the control torque, keeping the spacecraft pointed on target. In case this is undesirable, a penalty could be added to the reward for larger amplitudes in modal vibrations, to incentivize the agents to dampen the vibrations. In this case, full observability would be expected to further improve the performance of the agents, since the agents would have direct access to the states they need to dampen. In this research, however, this was outside of the scope.

The insights for the flexible model are hence used to formulate additional answers to the research sub-questions, specific to the flexible case:

- *SQ1: How can model-free reinforcement learning algorithms be effectively adapted for the design of a controller for post-capture control in the context of active debris removal missions?*
  - → In case system flexibilities are present, the deterministic algorithms TD3 and TD7 are able to deal with these adequately, resulting in performance better than a hand-tuned PD controller. PPO is not adequate for this case at all with the settings described in this research. SAC also deals with the flexible model effectively, but the performance is lacking with respect to the PD controller.
  - → SAC, TD3, and TD7 exhibit rapid and large oscillation behavior in the commanded control torque. In case this is not desirable, this might be mitigated with regularization techniques, and should be investigated further.
- *SQ2: To what extent is the stability and convergence of a model-free reinforcement learning-based controller reliable?*
  - → SAC and TD7 exhibit better robustness than TD3, however generally slightly worse than the PD controller. In general, the off-policy methods tried in this research seem to generalize relatively well, which for some perturbation types result in a tested convergence rate of 100%.
  - → In the flexible case, provable performance guarantees remain difficult or impossible to find.
- *SQ3: How do inherent uncertainties in the dynamic and kinematic properties of the system affect the performance of a model-free reinforcement learning-based controller in the post-capture phase of an active debris removal mission?*
  - → Between all the algorithms, the agents exhibit the ability to either dampen inertial vibrations, or compensate for these vibrations while keeping the spacecraft pointed to within the requirements.
  - → In case flexibilities are present in the system, RL agents can demonstrate performance significantly better than a hand-tuned PD controller, such as in the settling time. Hence, there is definitive potential for such agents to deal with the uncertainties in an end-to-end fashion. However, this is not always the case. For some of the perturbations the PD controller obtains a much lower settling time, such as for inertia rotation or torque misalignment.

Together with the insights from the rigid case, the answers to the sub-questions allow for the formulation of an answer to the main research question:

> To what extent can the implementation of model-free reinforcement learning algorithms impact the performance of post-capture control in active debris removal missions under the influence of inherent dynamic and kinematic uncertainties?

Off-policy model-free reinforcement learning algorithms can deliver better performance than a hand-tuned PD controller for the post-capture control in active debris removal missions under the influence of flexibilities in the system, imperfections in the torque actuation, and uncertainties in inertial model parameters. The on-policy algorithm PPO does not provide adequate performance. A clear difference between the performance of stochastic and deterministic policies was not observed.

Performance could possibly be improved further by tuning hyperparameters, but no straightforward relationships between hyperparameter values and agent performance have been identified. Furthermore, domain randomization might be an option for improving model performance, but implementing this correctly does not guarantee an improvement in agent performance and might even be detrimental to the performance. The RL agents are relatively robust, except for the case where the input to the

model is disturbed in the form of (relatively) large imperfections in the gyroscope measurements. In the end-to-end implementation in this research, SAC and TD7 already proved to be more robust than the hand-tuned PD controller in some cases, but did not always achieve better performance.

Finally, the control behavior of the agents in the flexible case is sometimes non-ideal as it oscillates between large positive and negative commanded torque values to compensate for the internal vibrations. Further approaches to damping these vibrations should likely be investigated before these algorithms will be applicable for post-capture attitude control of an active debris removal mission.

5.2. Recommendations

The insights in this research can be leveraged to design a controller optimized for a specific real mission. The uncertainties associated with such a mission can then also be quantified, as well as clear requirements on the robustness of the controller. This is the first recommendation of this research, to investigate how these results transfer to a real mission in the real world.

While the effects of domain randomization and hyperparameter optimization were investigated to a first order, the optimization can be done in a lot more detail. In this research it was found that some combinations of hyperparameters and settings would often result in failure of the algorithms to produce convergent agents, but explaining why remains difficult. Doing an in-depth optimization for a single algorithm is a good opportunity to unify the changes in performance with the theoretical background of the algorithms.

This optimization should also incorporate a more in-depth reliability study. In this research, the agents were simply tested for many initial conditions. The robustness could be tested to a much better extent. An interesting machine-learning based method could be to train an adversarial network that finds initial conditions or small changes/uncertainties to the model that the controlling agents fail for, and then co-training these agents. This could also be an interesting avenue to investigate the extent to which domain randomization can improve the performance of these agents, instead of the semi-quantitative method employed in this research.

A follow-up research should also focus on the ability of the reinforcement learning algorithms to fine-tune on-line. This potentially could make the agents much more adaptable to the perturbations, and is fundamentally one other big reason why reinforcement learning algorithms are an interesting study topic for application in post-capture control of an ADR mission in general, where adaptability could be very important.

Moreover, state estimators like advanced Kalman Filters or disturbance observers (even though they can be difficult to tune) should be looked at for implementation in the feedback loop to estimate disturbances and filter out parametric noise like gyroscope bias and noise, which might improve robustness significantly [146][147]. While the RL agents are not able to deal with these gyroscope perturbations in an end-to-end fashion, it could be expected that with the inclusion of such an estimator/observer, an RL-based controller could offer high performance.

The TD7 algorithm includes an addition for offline learning, which implements practically a form of imitation learning. This was not used in this research, due to the end-to-end training process investigation. Using this functionality for the initial few training steps, for example by letting the agent learn from the output of the PD controller, it could be a suitable method to even further improve the sample efficiency of this algorithm.

Furthermore, the full model has been validated thoroughly in this work. It is made to be very adaptable, requiring minimal adjustments to implement different algorithms, as long as the algorithms are written within the Stable-Baselines 3 API. Many such algorithms are publicly available, so it would be interesting and relatively easy to repeat the experiments with more algorithms.

Besides potential follow-on studies, the presented study can also be improved in several fundamental ways. The first is to have another look at the selection of domain randomization and perturbation types and especially their magnitudes. This would offer much deeper insight into the extent of the robustness of the tested controllers, instead of a qualitative measure of robustness. Furthermore, the effects were investigated only one at a time. The application of a combination of several kinds of randomizations at

the same time is usually the way domain randomization is best applied [80].

A further remark on the current research is that of sampling frequency and latency. The sampling frequency was chosen relatively arbitrarily, while effects of changing this (especially on the flexible model) were not quantified. The sampling frequency is an important metric to carefully set, especially if computational power is a limiting factor (as is likely the case when using deep feed-forward neural nets). The latency is the second factor of this that was neglected in this research. Delays in the feedback signal can greatly deteriorate reinforcement learning agent performance, such as shown by [148]. Furthermore, a non-zero latency is inevitable in a real-world implementation. This is hence an important factor to address in the design of RL agents for attitude control.

A small shortcoming of the research is the inconsistent training times and sample sizes. These decisions were made due to scarce computational resources, and because they were deemed to be reasonable compromises between provided confidence in the results and the amount of different experiments this enabled. In a continuation of the research, the agents should be trained to the full extent at least once per experiment, so that no new behavior later in the training gets overlooked. Furthermore, the sample size should be increased for the experiments with large standard deviation in the data (or experiments that often failed and as a results have few successful results to work with).

The flexibility analysis of this research also only remained more broad than deep. The effect of different model parameters than the chosen inertia, δ, K, and C (or even a higher-fidelity model) has not been investigated. Important in this case is the ability of the agents to deal with the excited vibrational modes. A more in-depth analysis could be done by analyzing the frequency response of the system with the trained agents as controller. An idea about the damping for a range of frequencies can be obtained using this method, which is useful for meeting and validating possible vibrational requirements. As discussed in section 5.1, a penalty for larger vibrations could be added to the reward function to incentivize the agents to dampen the vibrations. The effect of such penalties would be nicely quantified by testing the frequency response with different agents in the control loop.

A deeper improvement to the current research is to relate the flexible model parameters more to an actual mission with a robotic arm. For generalized research and semi-quantitative insights into the effects of flexibilities the current model sufficed. However, the link between the current model and real-world ADR spacecraft problems is relatively weak, and should be evaluated and further validated in future research. Examples of how this can be done are a finite element analysis [118] or a multibody analysis [149].

The PD controller is another limiting factor of this research. Even though it does provide a simple and intuitive baseline for comparison of the agents, the controller is not very advanced or representative of the serious alternatives that designers of attitude controllers for a post-capture phase in an ADR mission will likely consider. The first step would be to tune the PD (or even a full PID) in a more structured way manually or to use off-the-shelf software tools that do this automatically. A further step would be to take a representative controller from literature and to compare the agents against it.

Finally, this research demonstrated that in the more complex flexible model, some RL agents can exhibit very favorable properties, resulting in a significantly better performance than the PD controller. However, due to the absence of performance guarantees, it is difficult to see how a reinforcement learning agent can be reliably used in a real spacecraft. Prior research has aimed to combine the favorable properties of the RL agents with the reliability and explainability of a conventional controller. This could also be further investigated in follow-up research.

References

[1] Pierre Bernhard, Marc Deschamps, and Georges Zaccour. “Large satellite constellations and space debris: Exploratory analysis of strategic management of the space commons”. In: *European Journal of Operational Research* 304.3 (2023), pp. 1140–1157.

[2] Donald J. Kessler and Burton G. Cour-Palais. “Collision frequency of artificial satellites: The creation of a debris belt”. In: *Journal of Geophysical Research: Space Physics* 83.A6 (1978), pp. 2637–2646.

[3] K. Merz et al. “Current collision avoidance service by ESA’s Space Debris Office”. In: *7th European Conference on Space Debris*. 2017, p. 219.

[4] Jer Chyi Liou. “Highlights of recent research activities at the NASA Orbital Debris Program Office”. In: *European Conference on Space Debris*. JSC-CN-39199. 2017.

[5] Hanspeter Schaub et al. “Cost and risk assessment for spacecraft operation decisions caused by the space debris environment”. In: *Acta Astronautica* 113 (2015), pp. 66–79.

[6] Minghe Shan, Jian Guo, and Eberhard Gill. “Analysis of the Concept of Noncooperative Targets and Associated Tailored Active Debris Removal Methods”. In: *IAC Conference*. 2014.

[7] Phillip Anz-Meador, John Opiela, and Jer-Chyi Liou. *History of on-orbit satellite fragmentations*. Tech. rep. 2023.

[8] A. Rossi and G.B. Valsecchi. “Collision risk against space debris in Earth orbits”. In: *Celestial Mechanics and Dynamical Astronomy* 95.1-4 (2006), pp. 345–356.

[9] NASA Orbital Debris Program Office. *LEGEND: 3D/OD Evolutionary Model*. URL: `https://www.orbitaldebris.jsc.nasa.gov/modeling/legend.html`. (accessed: 30-08-2023).

[10] IADC Steering Group and Working Group 4. *IADC Space Debris Mitigation Guidelines*. URL: `https://iadc-home.org/documents_public/index`. (accessed: 30-08-2023).

[11] Jer Chyi Liou and Nicholas L. Johnson. “Instability of the present LEO satellite populations”. In: *Advances in Space Research* 41.7 (2008), pp. 1046–1053. ISSN: 0273-1177.

[12] B. Bastida and H. Krag. “Analyzing the criteria for a stable environment”. In: *AAS/AIAA Astrodynamics Specialist Conference, Girdwood, Alaska*. 2011.

[13] Jer Chyi Liou, Nicholas L. Johnson, and Nicole M. Hill. “Controlling the growth of future LEO debris populations with active debris removal”. In: *Acta Astronautica* 66.5-6 (2010), pp. 648–653.

[14] Remi Soulard et al. “ICAN: A novel laser architecture for space debris removal”. In: *Acta Astronautica* 105.1 (2014), pp. 192–200.

[15] Claudio Bombardelli and Jesus Peláez. “Ion beam shepherd for contactless space debris removal”. In: *Journal of guidance, control, and dynamics* 34.3 (2011), pp. 916–920.

[16] Minghe Shan, Jian Guo, and Eberhard Gill. “Review and comparison of active space debris capturing and removal methods”. In: *Progress in Aerospace Sciences* 80 (2016), pp. 18–32.

[17] Robin Biesbroek et al. “The e. deorbit CDF study: a design study for the safe removal of a large space debris”. In: *64th International Astronautical Congress (IAC), Beijing*. 2013.

[18] D. Reintsema et al. “DEOS–the German robotics approach to secure and de-orbit malfunctioned satellites from low earth orbits”. In: *Proceedings of the i-SAIRAS*. Japan Aerospace Exploration Agency (JAXA) Japan. 2010, pp. 244–251.

[19] Robin Biesbroek et al. “e. Deorbit-ESA’s active debris removal mission”. In: *Proceedings of the 7th European Conference on Space Debris*. Vol. 10. ESA Space Debris Office. 2017.

[20] Evangelos Papadopoulos et al. “Robotic manipulation and capture in space: A survey”. In: *Frontiers in Robotics and AI* (2021), p. 228.

[21] Guglielmo S. Aglietti et al. “RemoveDEBRIS: An in-orbit demonstration of technologies for the removal of space debris”. In: *The Aeronautical Journal* 124.1271 (2020), pp. 1–23.

[22] Guglielmo S. Aglietti et al. “The active space debris removal mission RemoveDebris. Part 2: In orbit operations”. In: *Acta Astronautica* 168 (2020), pp. 310–322.

[23] Jason Forshaw et al. “The ELSA-d End-of-life debris removal mission: mission design, in-flight safety, and preparations for launch”. In: *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference, Maui Economic Development Board, Kihei, HI, USA*. 2019, pp. 17–20.

[24] Aishling Dignam et al. “In-Space Situational Awareness: Developing Spaceborne Sensors for Detecting, Tracking and Characterising Space Debris”. In: *2nd NEO and Debris Detection Conference*. 2023, p. 50.

[25] Robin Biesbroek et al. “The clearspace-1 mission: ESA and clearspace team up to remove debris”. In: *Proc. 8th Eur. Conf. Sp. Debris*. 2021, pp. 1–3.

[26] Christophe Bonnal, Jean-Marc Ruault, and Marie-Christine Desjean. “Active debris removal: Recent progress and current trends”. In: *Acta Astronautica* 85 (2013), pp. 51–60.

[27] Jer Chyi Liou. “An active debris removal parametric study for LEO environment remediation”. In: *Advances in space research* 47.11 (2011), pp. 1865–1876.

[28] C. Wiedemann et al. “Cost estimation of active debris removal”. In: *Proceedings of the International Astronautical Congress, IAC* 4 (Jan. 2012), pp. 2637–2646.

[29] Paula S. Morgan. “Fault protection techniques in JPL Spacecraft”. In: (2005).

[30] Christopher Masaru Pong. “High-precision pointing and attitude estimation and control algorithms for hardware-constrained spacecraft”. PhD thesis. Massachusetts Institute of Technology, 2014.

[31] Valdemir Carrara et al. “The ITASAT cubesat development and design”. In: *Journal of Aerospace Technology and Management* 9 (2017), pp. 147–156.

[32] Jürgen Telaar et al. “GNC architecture for the e. Deorbit mission”. In: *7th European Conference for Aeronautics and Space Sciences (EUCASS)*. ESA Publications Division, ESTEC Noordwijk, The Netherlands. 2017, pp. 1–15.

[33] Pablo Colmenarejo et al. “Results of the COMRADE project: combined control for robotic spacecraft and manipulator in servicing missions: active debris removal and re-fuelling”. In: *11th International ESA Conference on Guidance, Navigation & Control Systems*. 2020, pp. 10–23.

[34] C.M. Pong and D.W. Miller. “Angular rate estimation from geomagnetic field measurements and observability singularity avoidance during detumbling and sun acquisition”. In: *AAS Guidance and Control Conference, Breckenridge, CO*. 2013.

[35] Panfeng Huang et al. “Attitude takeover control for post-capture of target spacecraft using space robot”. In: *Aerospace Science and Technology* 51 (2016), pp. 171–180.

[36] G.A. Beals et al. “Hubble Space Telescope precision pointing control system”. In: *Journal of Guidance, Control, and Dynamics* 11.2 (1988), pp. 119–123.

[37] Panfeng Huang et al. “Reconfigurable spacecraft attitude takeover control in post-capture of target by space manipulators”. In: *Journal of the Franklin Institute* 353.9 (2016), pp. 1985–2008.

[38] Sunayna Singh and Erwin Mooij. “Robust control for active debris removal of a large flexible space structure”. In: *AIAA Scitech 2020 Forum*. 2020, p. 2077.

[39] Zheng Wang, Jianping Yuan, and Dejia Che. “Adaptive attitude takeover control for space non-cooperative targets with stochastic actuator faults”. In: *Optik* 137 (2017), pp. 279–290.

[40] Kai Ning, Baolin Wu, and Chuang Xu. “Event-triggered adaptive fuzzy attitude takeover control of spacecraft”. In: *Advances in Space Research* 67.6 (2021), pp. 1761–1772.

[41] Henzeh Leeghim, Yoonhyuk Choi, and Hyochoong Bang. “Adaptive attitude control of spacecraft using neural networks”. In: *Acta Astronautica* 64.7-8 (2009), pp. 778–786.

[42] Caisheng Wei et al. “Learning-based adaptive prescribed performance control of postcapture space robot-target combination without inertia identifications”. In: *Acta Astronautica* 146 (2018), pp. 228–242.

[43] Xiuwei Huang, James D. Biggs, and Guangren Duan. “Post-capture attitude control with prescribed performance”. In: *Aerospace Science and Technology* 96 (2020), p. 105572.

[44] Jianjun Luo et al. “Robust inertia-free attitude takeover control of postcapture combined spacecraft with guaranteed prescribed performance”. In: *ISA transactions* 74 (2018), pp. 28–44.

[45] Xiuwei Huang and Guangren Duan. “Fault-tolerant attitude tracking control of combined spacecraft with reaction wheels under prescribed performance”. In: *ISA transactions* 98 (2020), pp. 161–172.

[46] Han Gao et al. “Forecasting-based data-driven model-free adaptive sliding mode attitude control of combined spacecraft”. In: *Aerospace Science and Technology* 86 (2019), pp. 364–374.

[47] Panfeng Huang et al. “Postcapture attitude takeover control of a partially failed spacecraft with parametric uncertainties”. In: *IEEE Transactions on Automation Science and Engineering* 16.2 (2018), pp. 919–930.

[48] Panfeng Huang et al. “Adaptive postcapture backstepping control for tumbling tethered space robot–target combination”. In: *Journal of Guidance, Control, and Dynamics* 39.1 (2016), pp. 150–156.

[49] Jean-Jacques E. Slotine, Weiping Li, et al. *Applied nonlinear control*. Vol. 199. 1. Prentice hall Englewood Cliffs, NJ, 1991.

[50] Barton Bacon and Aaron Ostroff. “Reconfigurable flight control using nonlinear dynamic inversion with a special accelerometer implementation”. In: *AIAA Guidance, Navigation, and Control Conference and Exhibit*. 2000, p. 4565.

[51] Paul Acquatella et al. “Robust Nonlinear Spacecraft Attitude Control using Incremental Nonlinear Dynamic Inversion.” In: *AIAA Guidance, Navigation, and Control Conference*. 2012, p. 4623.

[52] Ronald van’t Veld, Erik-Jan van Kampen, and Qi Ping Chu. “Stability and robustness analysis and improvements for incremental nonlinear dynamic inversion control”. In: *2018 AIAA Guidance, Navigation, and Control Conference*. 2018, p. 1127.

[53] Xuerui Wang et al. “Stability analysis for incremental nonlinear dynamic inversion control”. In: *Journal of Guidance, Control, and Dynamics* 42.5 (2019), pp. 1116–1129.

[54] Ye Zhou, Erik-Jan Van Kampen, and Qi Ping Chu. “Incremental model based online heuristic dynamic programming for nonlinear adaptive tracking control with partial observability”. In: *Aerospace Science and Technology* 105 (2020), p. 106013.

[55] Zhi-Gang Zhou et al. “Adaptive actor-critic learning-based robust appointed-time attitude tracking control for uncertain rigid spacecrafts with performance and input constraints”. In: *Advances in Space Research* 71.9 (2023), pp. 3574–3587.

[56] Anh Tuan Vo, Thanh Nguyen Truong, and Hee-Jun Kang. “Fixed-time rbfnn-based prescribed performance control for robot manipulators: Achieving global convergence and control performance improvement”. In: *Mathematics* 11.10 (2023), p. 2307.

[57] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[58] Yuhan Liu et al. “Neural network-based reinforcement learning control for combined spacecraft attitude tracking maneuvers”. In: *Neurocomputing* 484 (2022), pp. 67–78.

[59] Gabriel Dulac-Arnold et al. “Challenges of real-world reinforcement learning: definitions, benchmarks and analysis”. In: *Machine Learning* 110.9 (2021), pp. 2419–2468.

[60] Dmitry Kalashnikov et al. “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *arXiv preprint arXiv:1806.10293* (2018).

[61] Mel Vecerik et al. “A practical approach to insertion with variable socket position using deep reinforcement learning”. In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 754–760.

[62] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. “Sim-to-real transfer in deep reinforcement learning for robotics: a survey”. In: *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE. 2020, pp. 737–744.

[63] Joshua Achiam. *Spinning Up in Deep Reinforcement Learning*. URL: `https://github.com/openai/spinningup`. (accessed: 12-11-2023).

[64] Richard Bellman. “Some problems in the theory of dynamic programming”. In: *Econometrica: Journal of the Econometric Society* (1954), pp. 37–48.

[65] Richard Bellman. “A Markovian decision process”. In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[66] Ronald A. Howard. “Dynamic programming and markov processes.” In: (1960).

[67] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine learning* 3 (1988), pp. 9–44.

[68] John N. Tsitsiklis and Benjamin Van Roy. “An analysis of temporal-difference learning with function approximation”. In: *Rep. LIDS-P-2322). Lab. Inf. Decis. Syst. Massachusetts Inst. Technol. Tech. Rep* (1996).

[69] Andrew G. Barto and Sridhar Mahadevan. “Recent advances in hierarchical reinforcement learning”. In: *Discrete event dynamic systems* 13.1-2 (2003), pp. 41–77.

[70] Yang Yu. “Towards Sample Efficient Reinforcement Learning.” In: *IJCAI*. 2018, pp. 5739–5743.

[71] Lasse Espeholt et al. “Seed rl: Scalable and efficient deep-rl with accelerated central inference”. In: *arXiv preprint arXiv:1910.06591* (2019).

[72] Ye Zhou, Erik-Jan van Kampen, and Qi Ping Chu. “Adaptive spacecraft attitude control with incremental approximate dynamic programming”. In: *68th International Astronautical Congress (IAC)*. 2017, pp. 25–29.

[73] Peter Dayan and Christopher J.C.H. Watkins. “Q-learning”. In: *Machine learning* 8.3 (1992), pp. 279–292.

[74] Daniel Neider et al. “Advice-guided reinforcement learning in a non-markovian environment”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 10. 2021, pp. 9073–9080.

[75] Rania Hassan and William Crossley. “Spacecraft reliability-based design optimization under uncertainty including discrete variables”. In: *Journal of Spacecraft and Rockets* 45.2 (2008), pp. 394–405.

[76] Nicholay Topin and Manuela Veloso. “Generation of policy-level explanations for reinforcement learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 2514–2521.

[77] Minghao Han et al. “Reinforcement learning control of constrained dynamic systems with uniformly ultimate boundedness stability guarantee”. In: *Automatica* 129 (2021), p. 109689.

[78] Hado Van Hasselt et al. “Deep reinforcement learning and the deadly triad”. In: *arXiv preprint arXiv:1812.02648* (2018).

[79] Qingyan Huang. “Model-based or model-free, a review of approaches in reinforcement learning”. In: *2020 International Conference on Computing and Data Science (CDS)*. IEEE. 2020, pp. 219–221.

[80] OpenAI: Marcin Andrychowicz et al. “Learning dexterous in-hand manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[81] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. In: (1989).

[82] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[83] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[84] Yan Duan et al. “Benchmarking deep reinforcement learning for continuous control”. In: *International conference on machine learning*. PMLR. 2016, pp. 1329–1338.

[85] Yuejiao Wang et al. “A new spacecraft attitude stabilization mechanism using deep reinforcement learning method”. In: *8th European Conference for Aeronautics and Space Sciences (EUCASS)*. 2019.

[86] Richard S. Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems* 12 (1999).

[87] A. Rupam Mahmood et al. “Benchmarking reinforcement learning algorithms on real-world robots”. In: *Conference on robot learning*. PMLR. 2018, pp. 561–591.

[88] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[89] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[90] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[91] Eivind Bøhn et al. “Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization”. In: *2019 international conference on unmanned aircraft systems (ICUAS)*. IEEE. 2019, pp. 523–533.

[92] Jacob Elkins, Rohan Sood, and Clemens Rumpf. “Autonomous spacecraft attitude control using deep reinforcement learning”. In: *71st International Astronautical Congress (IAC)*. Vol. 2020. 2020.

[93] Jacob G Elkins, Rohan Sood, and Clemens Rumpf. “Bridging reinforcement learning and online learning for spacecraft attitude control”. In: *Journal of Aerospace Information Systems* 19.1 (2022), pp. 62–69.

[94] Nessrine Hammami and Kim Khoa Nguyen. “On-policy vs. off-policy deep reinforcement learning for resource allocation in open radio access network”. In: *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2022, pp. 1461–1466.

[95] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[96] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[97] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[98] Jacob Elkins, Rohan Sood, and Clemens Rumpf. “Adaptive continuous control of spacecraft attitude using deep reinforcement learning”. In: *2020 AAS/AIAA Astrodynamics Specialist Conference*. AIAA Reston, VA. 2020, pp. 420–475.

[99] Scott Fujimoto et al. “For SALE: State-Action Representation Learning for Deep Reinforcement Learning”. In: *arXiv preprint arXiv:2306.02451* (2023).

[100] Scott Fujimoto, David Meger, and Doina Precup. “An equivalence between loss functions and non-uniform sampling in experience replay”. In: *Advances in neural information processing systems* 33 (2020), pp. 14219–14230.

[101] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).

[102] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[103] Tuomas Haarnoja et al. “Soft actor-critic algorithms and applications”. In: *arXiv preprint arXiv:1812.05905* (2018).

[104] Arsenii Kuznetsov et al. “Controlling overestimation bias with truncated mixture of continuous distributional quantile critics”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5556–5566.

[105] Jingliang Duan et al. “Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors”. In: *IEEE transactions on neural networks and learning systems* 33.11 (2021), pp. 6584–6598.

[106] Jacob Beck et al. “A survey of meta-reinforcement learning”. In: *arXiv preprint arXiv:2301.08028* (2023).

[107] Dhruv Ramani. “A short survey on memory based reinforcement learning”. In: *arXiv preprint arXiv:1904.06736* (2019).

[108] Jingkang Wang, Yang Liu, and Bo Li. “Reinforcement learning with perturbed rewards”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 04. 2020, pp. 6202–6209.

[109] Tom Everitt et al. “Reinforcement learning with a corrupted reward channel”. In: *arXiv preprint arXiv:1705.08417* (2017).

[110] F. Landis Markley and John L. Crassidis. *Fundamentals of spacecraft attitude determination and control*. Vol. 1286. Springer, 2014.

[111] Pedro Arroz and Fernando Gandía. *Detumbling Executive Summary*. Version 1.0. ESA Contract No. 4000113022/14/NL/MV. Isaac Newton, 11; PTM Tres Cantos; Madrid 28760, Sept. 2017. URL: www.gmv.com.

[112] B. Bastida Virgili, S. Lemmens, and H. Krag. “Investigation on Envisat Attitude Motion”. In: *e.Deorbit Workshop*. ESA/ESOC Space Debris Office, May 2014.

[113] Peter W Likins. *Dynamics and control of flexible space vehicles*. Tech. rep. 1970.

[114] S Monaco, DN Cyrot, and S Stornelli. “Sampled nonlinear control for large angle maneuvers of flexible spacecraft”. In: *ESA Proceedings of the Second International Symposium on Spacecraft Flight Dynamics p 31-38(SEE N 87-25354 19-18)*. 1986.

[115] Stefano Di Gennaro. “Passive attitude control of flexible spacecraft from quaternion measurements”. In: *Journal of optimization theory and applications* 116 (2003), pp. 41–60.

[116] Stefano Di Gennaro. “Output stabilization of flexible spacecraft with active vibration suppression”. In: *IEEE Transactions on Aerospace and Electronic systems* 39.3 (2003), pp. 747–759.

[117] S Di Gennaro. “Active vibration suppression in flexible spacecraft attitude tracking”. In: *Journal of Guidance, Control, and Dynamics* 21.3 (1998), pp. 400–408.

[118] Pierangela Morga, Mauro Mancini, and Elisa Capello. “Flexible spacecraft model and robust control techniques for attitude maneuvers”. In: *2022 American Control Conference (ACC)*. IEEE. 2022, pp. 1120–1126.

[119] Anne-Marie Lanouette et al. “Residual mechanical properties of a carbon fibers/PEEK space robotic arm after simulated orbital debris impact”. In: *International Journal of Impact Engineering* 84 (2015), pp. 78–87.

[120] Martijn Van Otterlo and Marco Wiering. “Reinforcement learning and markov decision processes”. In: *Reinforcement learning: State-of-the-art*. Springer, 2012, pp. 3–42.

[121] Yujing Hu et al. “Learning to utilize shaping rewards: A new approach of reward shaping”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15931–15941.

[122] Rodrigo Toro Icarte et al. “Reward machines: Exploiting reward function structure in reinforcement learning”. In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 173–208.

[123] Jan-Maarten Engel and Robert Babuška. “On-line reinforcement learning for nonlinear motion control: Quadratic and non-quadratic reward functions”. In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 7043–7048.

[124] Takuya Akiba et al. “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[125] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[126] Antonin Raffin. *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo. 2020.

[127] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[128] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[129] Fan Zhang et al. “On-line estimation of inertia parameters of space debris for its tether-assisted removal”. In: *Acta astronautica* 107 (2015), pp. 150–162.

[130] J Thienel and Robert M Sanner. “A coupled nonlinear spacecraft attitude controller and observer with an unknown constant gyro bias and gyro noise”. In: *IEEE transactions on Automatic Control* 48.11 (2003), pp. 2011–2015.

[131] Jovan D Boskovic, S-M Li, and Raman K Mehra. “Fault tolerant control of spacecraft in the presence of sensor bias”. In: *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*. Vol. 2. IEEE. 2000, pp. 1205–1209.

[132] KS Low, ST Goh, et al. “On-orbit gyroscope bias compensation to improve satellite attitude control performance”. In: *2021 IEEE Aerospace Conference (50100)*. IEEE. 2021, pp. 1–10.

[133] Tim Visser et al. “Torque model verification for the GOCE satellite”. In: *Advances in Space Research* 62.5 (2018), pp. 1114–1136.

[134] Xibin Cao et al. “Time efficient spacecraft maneuver using constrained torque distribution”. In: *Acta Astronautica* 123 (2016), pp. 320–329.

[135] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[136] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[137] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[138] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[139] Mark Towers et al. *Gymnasium*. Mar. 2023. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (visited on 07/08/2023).

[140] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: A llvm-based python jit compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1–6.

[141] University of Michigan. *Control Tutorials for MATLAB and Simulink - Introduction: PID Controller Design*. https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID. Accessed: 2024-05-31. 2024.

[142] James R Wertz. *Spacecraft attitude determination and control*. Vol. 73. Springer Science & Business Media, 2012.

[143] Behnam Neyshabur. “Implicit regularization in deep learning”. In: *arXiv preprint arXiv:1709.01953* (2017).

[144] Kamyar Azizzadenesheli, Yisong Yue, and Animashree Anandkumar. “Policy gradient in partially observable environments: Approximation and convergence”. In: *arXiv preprint arXiv:1810.07900* (2018).

[145] Douglas Aberdeen et al. “Policy-gradient algorithms for partially observable Markov decision processes”. In: (2003).

[146] Naeem Khan, M Irfan Khattak, and Dawei Gu. “Robust state estimation and its application to spacecraft control”. In: *Automatica* 48.12 (2012), pp. 3142–3150.

[147] Liang Sun and Zewei Zheng. “Disturbance-observer-based robust backstepping attitude stabilization of spacecraft under input saturation and measurement uncertainty”. In: *IEEE Transactions on Industrial Electronics* 64.10 (2017), pp. 7994–8002.

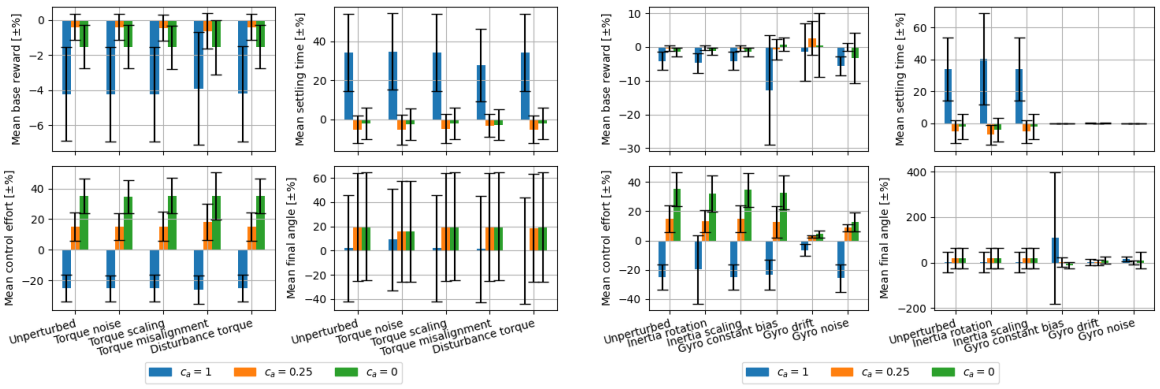
[148] Wei Wang et al. “Addressing Signal Delay in Deep Reinforcement Learning”. In: *The Twelfth International Conference on Learning Representations*. 2023.

[149] Sunayna Singh. “Multibody Approach to Controlled Removal of Large Space Debris with Flexible Appendages”. An electronic version of this thesis is available at `http://repository.tudelft.nl/`. MSc Thesis Report. Delft University of Technology, 2018.

A

Sensitivity analysis figures

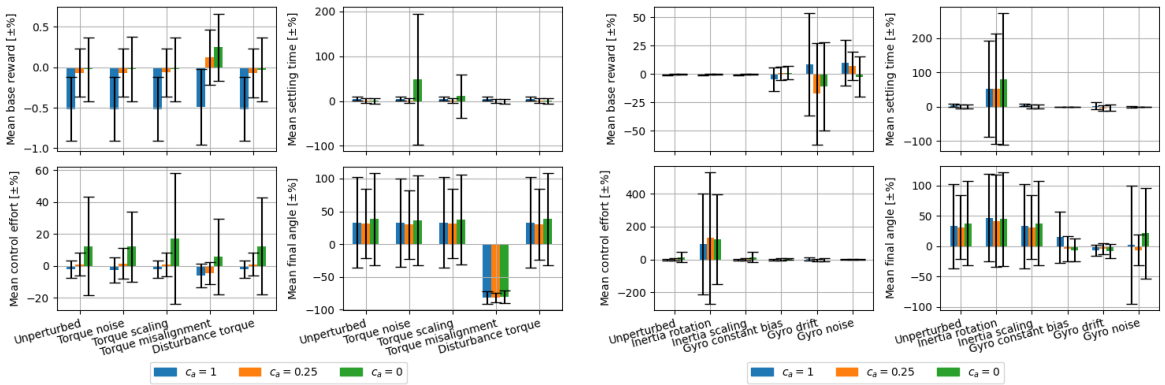
A.1. Reward function coefficients - rigid



(a) PPO - Torque perturbations

(b) PPO - Inertial and gyroscope perturbations

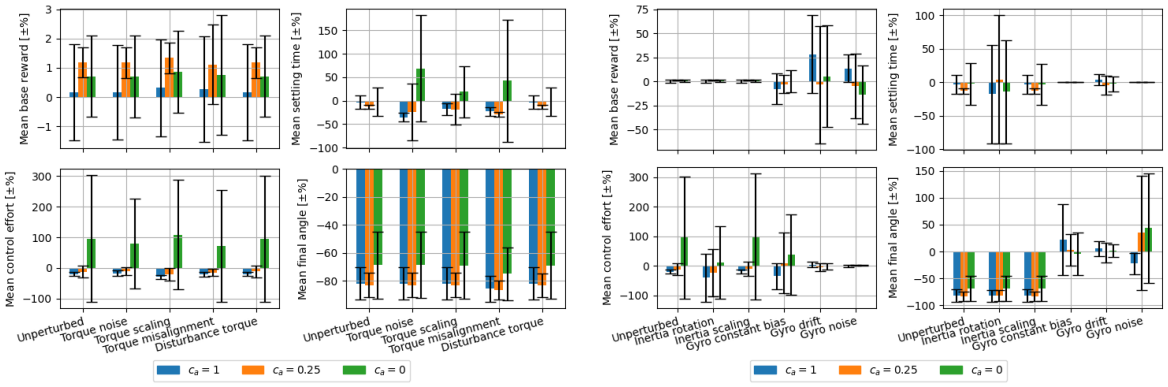
Figure A.1: Performance of PPO agents trained with variations in c_a in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.



(a) SAC - Torque perturbations

(b) SAC - Inertial and gyroscope perturbations

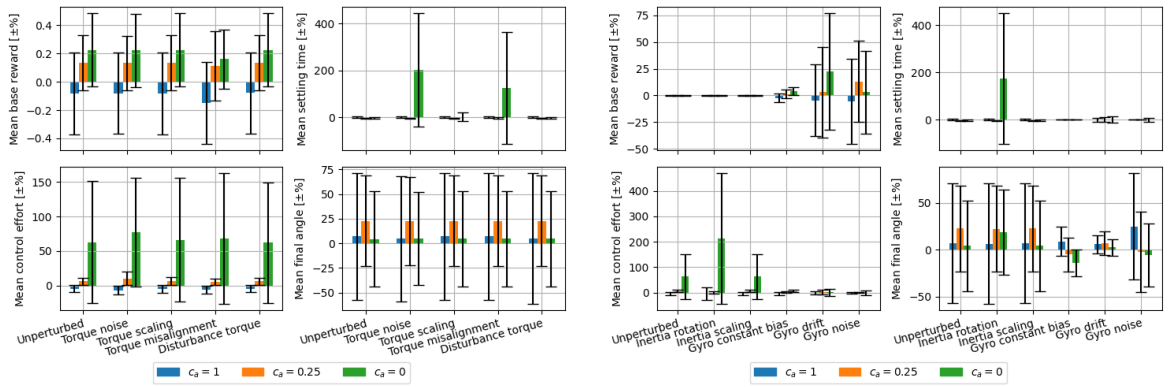
Figure A.2: Performance of SAC agents trained with variations in c_a in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.



(a) TD3 - Torque perturbations

(b) TD3 - Inertial and gyroscope perturbations

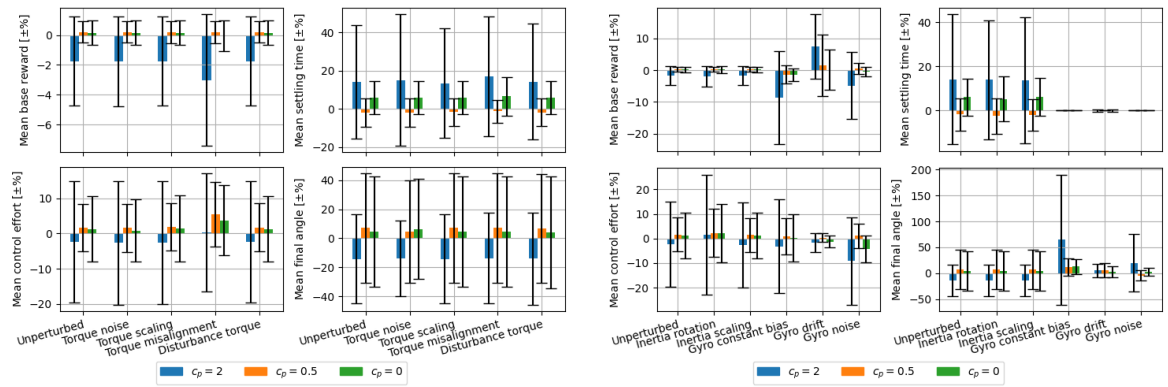
Figure A.3: Performance of TD3 agents trained with variations in c_a in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.



(a) TD7 - Torque perturbations

(b) TD7 - Inertial and gyroscope perturbations

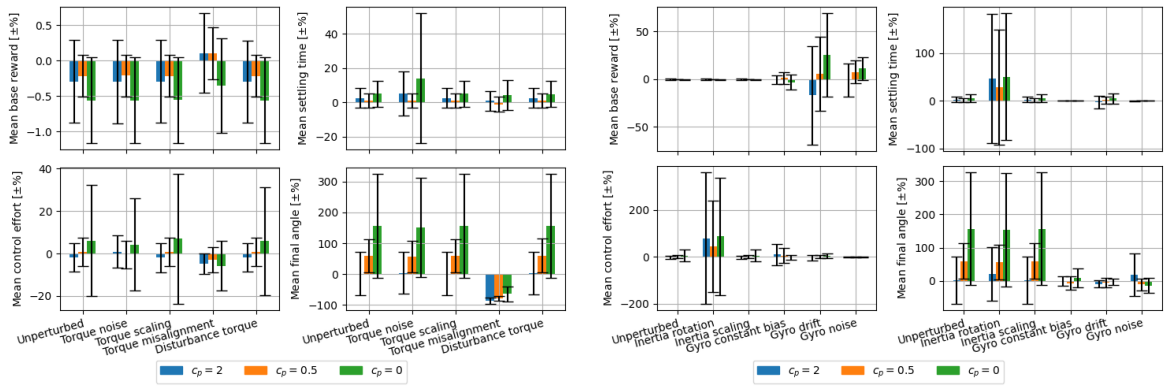
Figure A.4: Performance of TD7 agents trained with variations in c_a in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.



(a) PPO - Torque perturbations

(b) PPO - Inertial and gyroscope perturbations

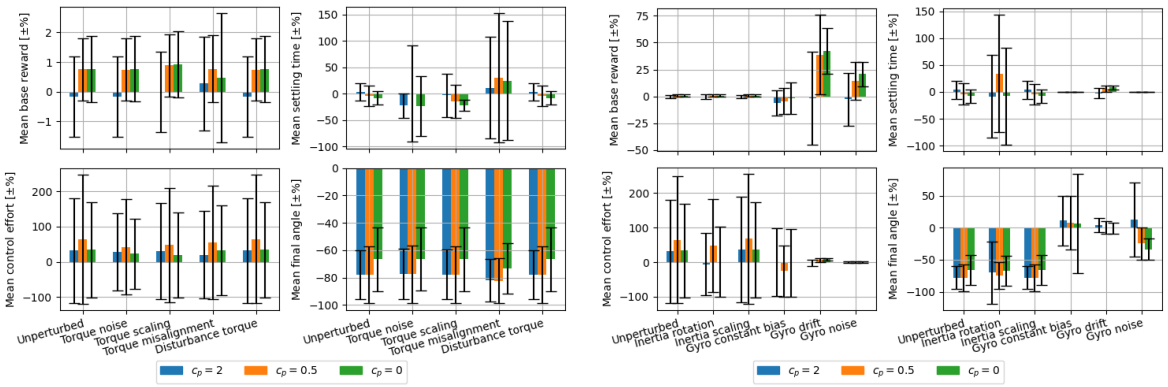
Figure A.5: Performance of PPO agents trained with variations in c_p in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.



(a) SAC - Torque perturbations

(b) SAC - Inertial and gyroscope perturbations

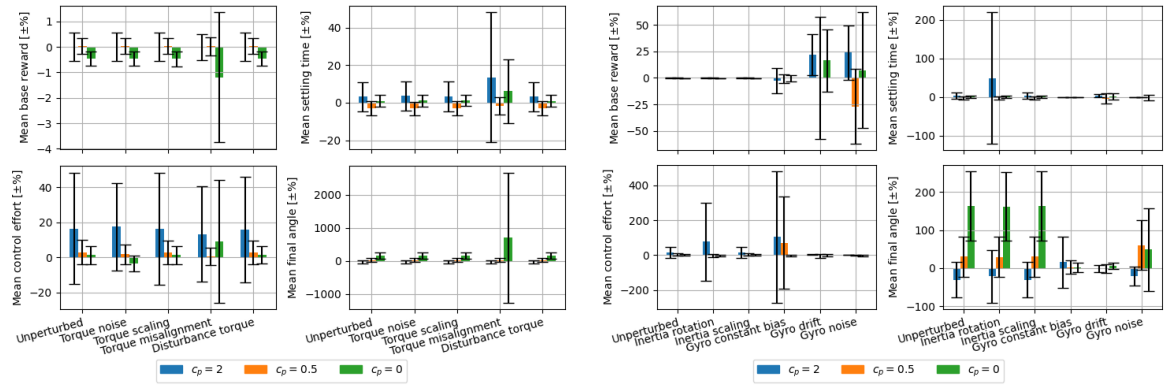
Figure A.6: Performance of SAC agents trained with variations in c_p in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.



(a) TD3 - Torque perturbations

(b) TD3 - Inertial and gyroscope perturbations

Figure A.7: Performance of TD3 agents trained with variations in c_p in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.



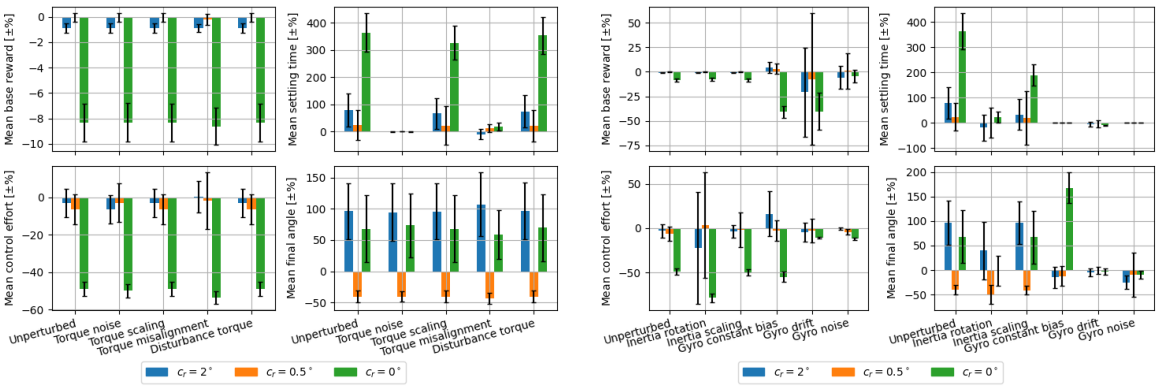
(a) TD7 - Torque perturbations

(b) TD7 - Inertial and gyroscope perturbations

Figure A.8: Performance of TD7 agents trained with variations in c_p in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the rigid environment. Bars indicate one standard deviation.

A.2. Reward function coefficients - flexible

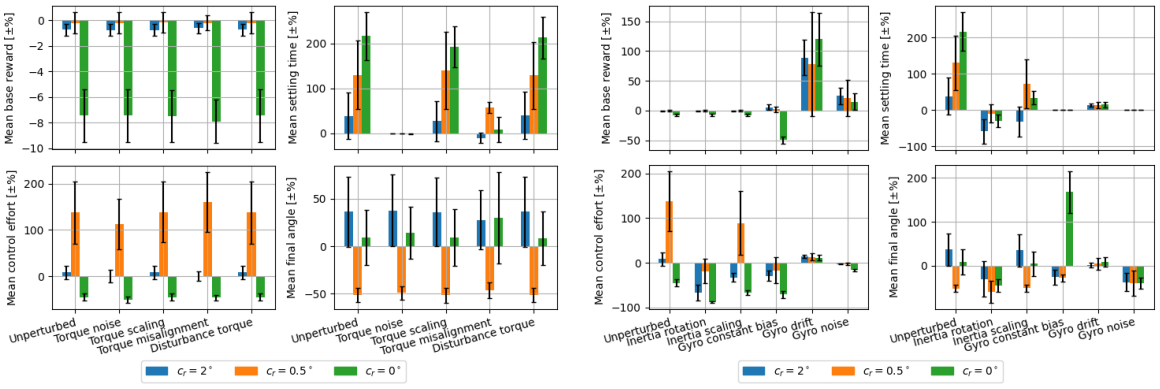
PPO did not result in any convergent agents for c_r, c_a, and c_p.



(a) SAC - Torque perturbations

(b) SAC - Inertial and gyroscope perturbations

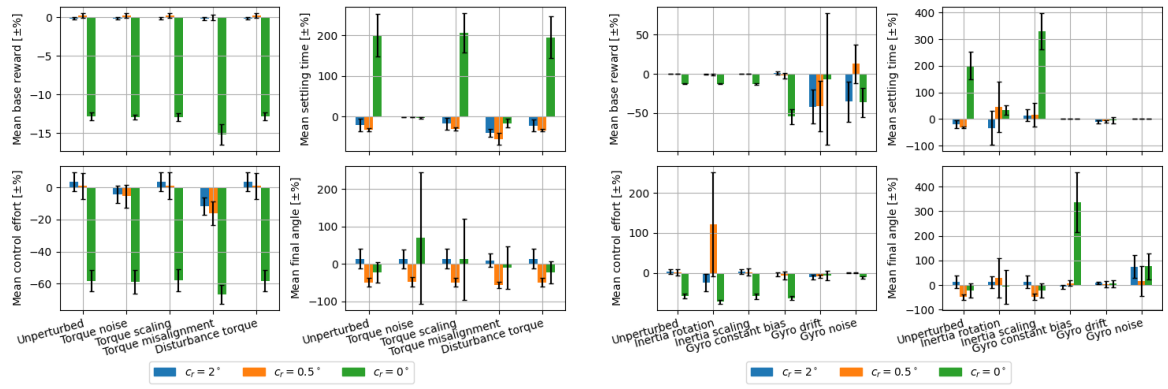
Figure A.9: Performance of SAC agents trained with variations in c_r in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.



(a) TD3 - Torque perturbations

(b) TD3 - Inertial and gyroscope perturbations

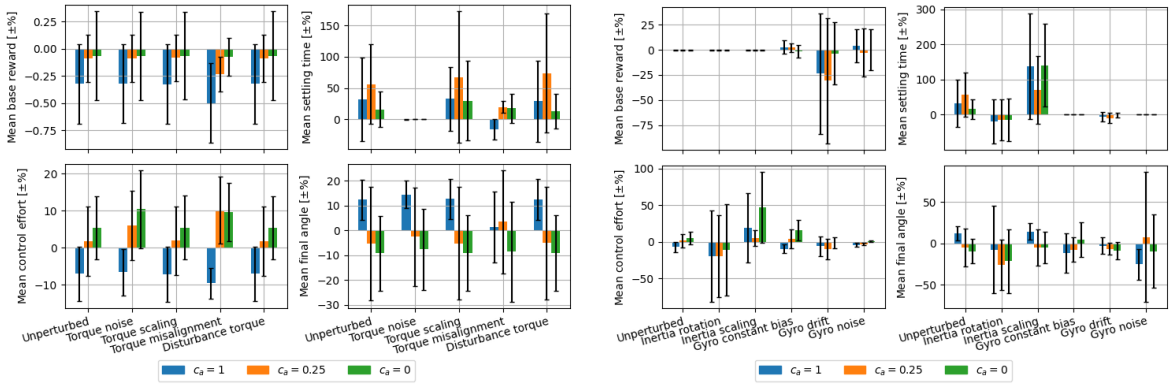
Figure A.10: Performance of TD3 agents trained with variations in c_r in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.



(a) TD7 - Torque perturbations

(b) TD7 - Inertial and gyroscope perturbations

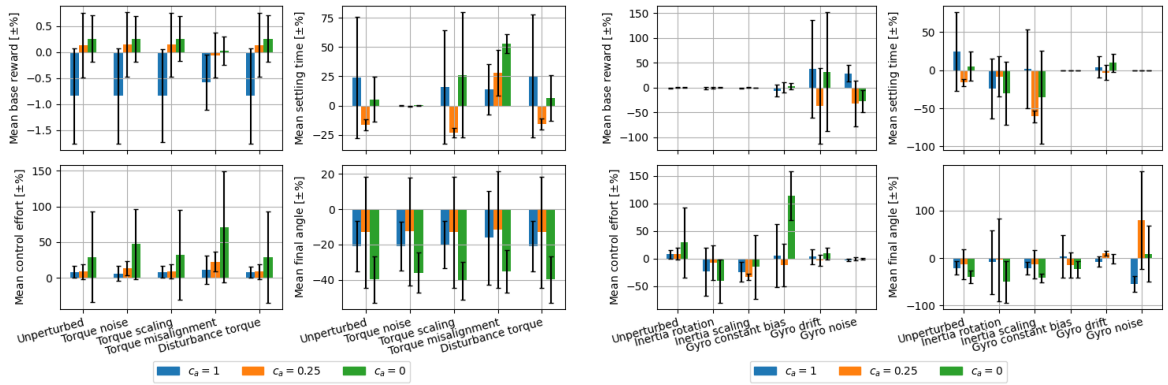
Figure A.11: Performance of TD7 agents trained with variations in c_r in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.



(a) SAC - Torque perturbations

(b) SAC - Inertial and gyroscope perturbations

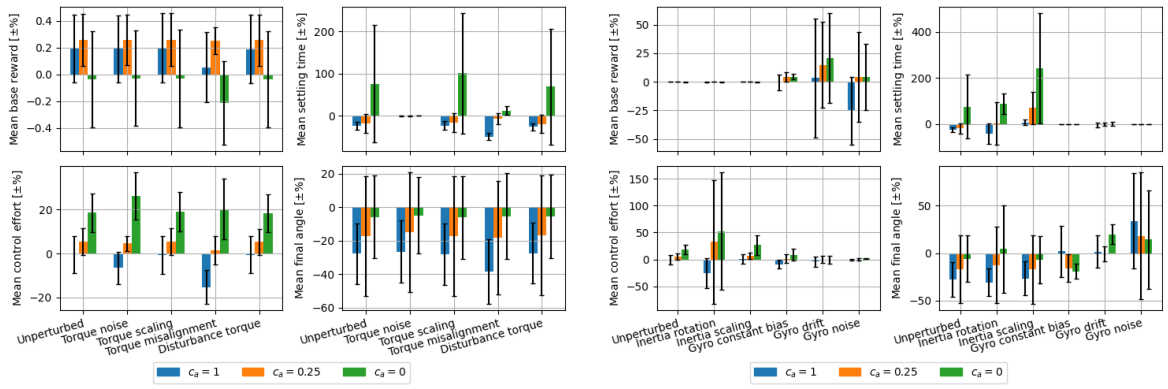
Figure A.12: Performance of SAC agents trained with variations in c_a in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.



(a) TD3 - Torque perturbations

(b) TD3 - Inertial and gyroscope perturbations

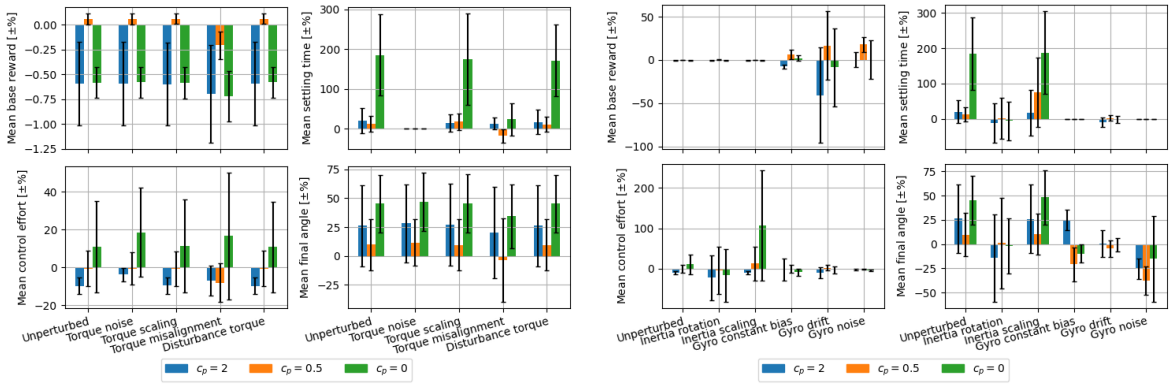
Figure A.13: Performance of TD3 agents trained with variations in c_a in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.



(a) TD7 - Torque perturbations

(b) TD7 - Inertial and gyroscope perturbations

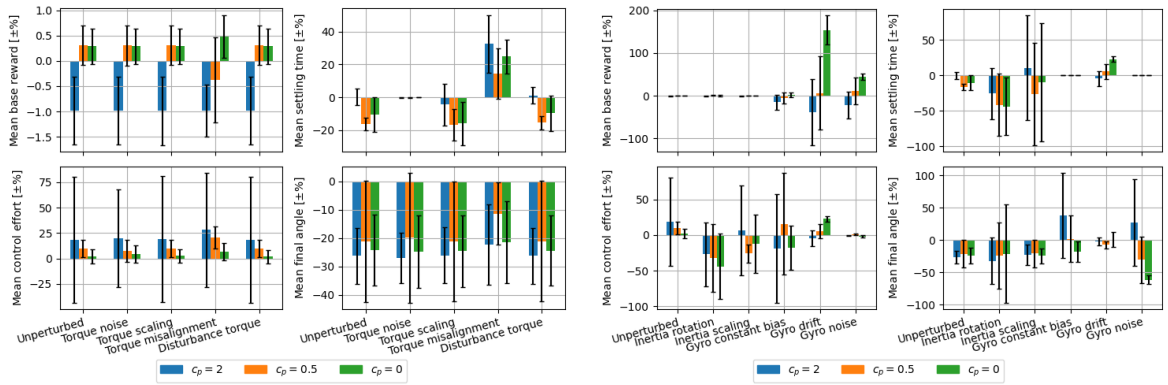
Figure A.14: Performance of TD7 agents trained with variations in c_a in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.



(a) SAC - Torque perturbations

(b) SAC - Inertial and gyroscope perturbations

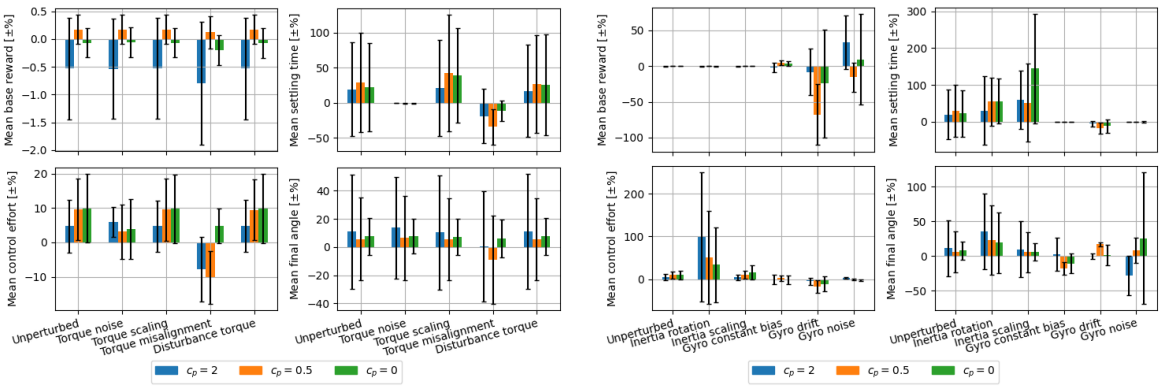
Figure A.15: Performance of SAC agents trained with variations in c_p in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.



(a) TD3 - Torque perturbations

(b) TD3 - Inertial and gyroscope perturbations

Figure A.16: Performance of TD3 agents trained with variations in c_p in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.



(a) TD7 - Torque perturbations

(b) TD7 - Inertial and gyroscope perturbations

Figure A.17: Performance of TD7 agents trained with variations in c_p in the reward function. Shown are the changes in mean performance with respect to the baseline, also under torque, inertial, and gyroscope perturbations, only for the flexible environment. Bars indicate one standard deviation.

A.3. Extra domain randomization results - rigid

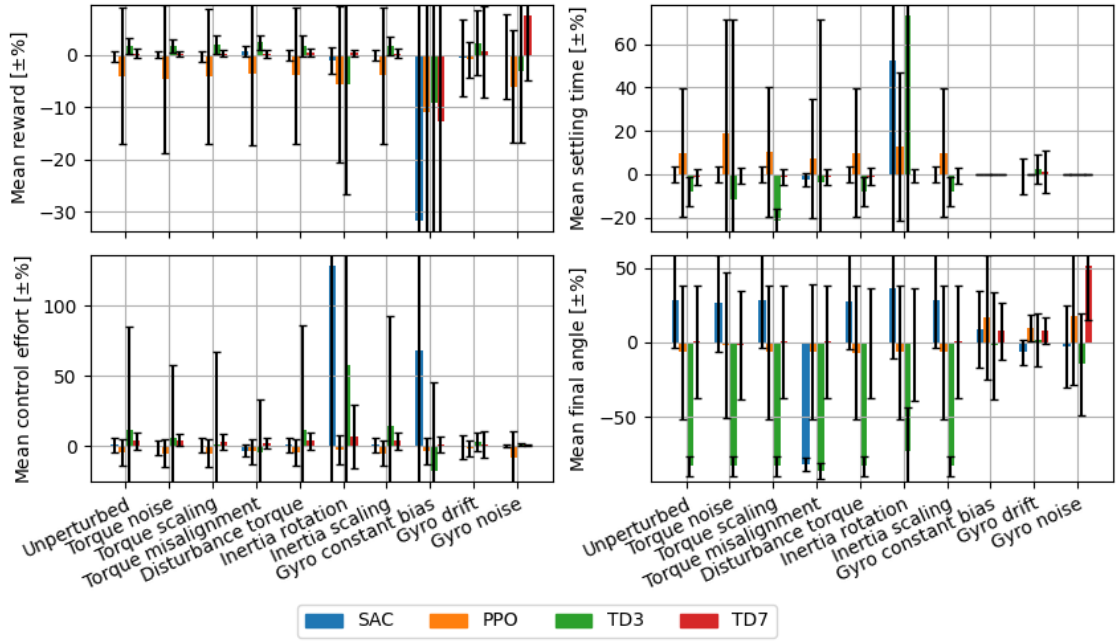


Figure A.18: Performance of agents with domain randomization applied in the form of inertia scaling, with respect to the baseline performance. Bars indicate one standard deviation.

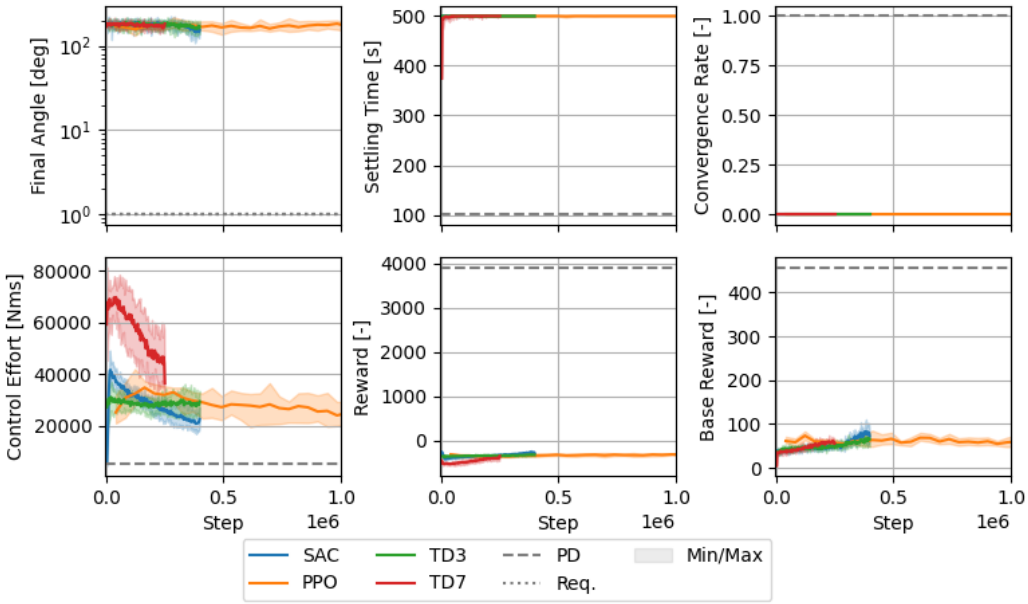


Figure A.19: Results of the learning process of the four different algorithms in the rigid spacecraft environment, with a drifting gyroscope bias domain randomization applied. Average results are shown including the minimum and maximum results, without any smoothing.

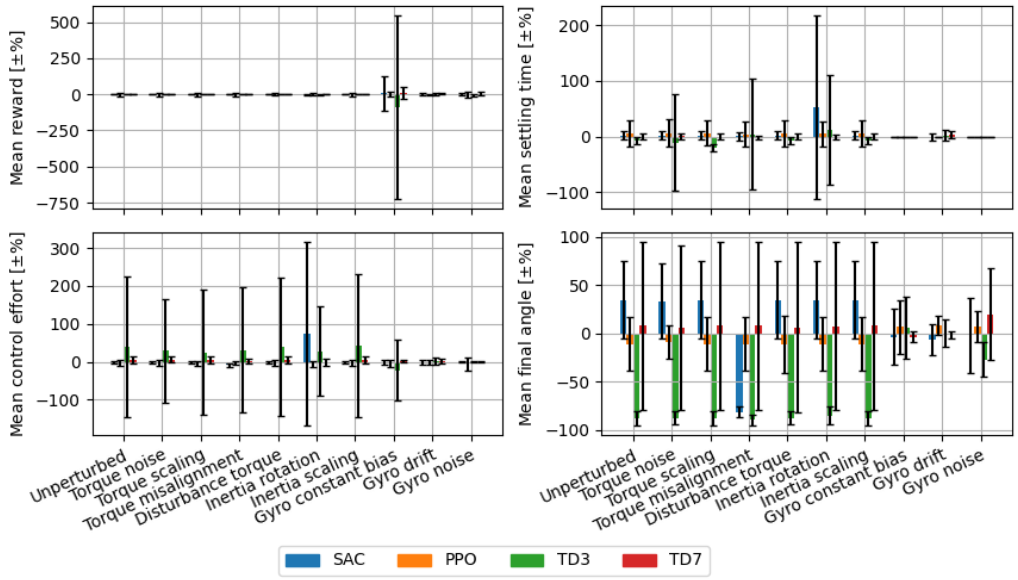


Figure A.20: Performance of agents with domain randomization applied in the form of torque scaling, with respect to the baseline performance in the rigid case. Bars indicate one standard deviation.

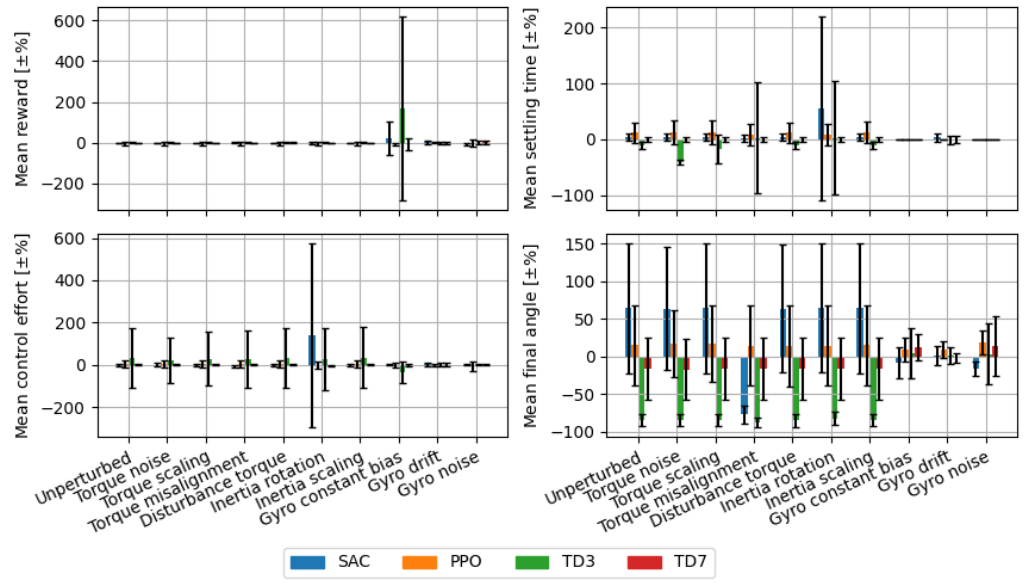


Figure A.21: Performance of agents with domain randomization applied in the form of torque noise, with respect to the baseline performance in the rigid case. Bars indicate one standard deviation.

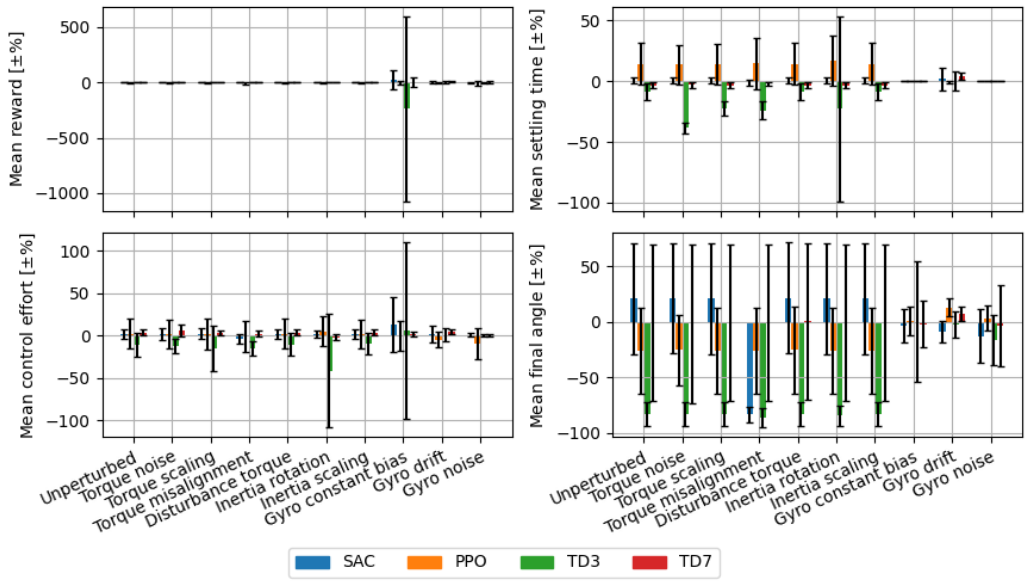


Figure A.22: Performance of agents with domain randomization applied in the form of a disturbance torque, with respect to the baseline performance in the rigid case. Bars indicate one standard deviation.

A.4. Extra domain randomization results - flexible

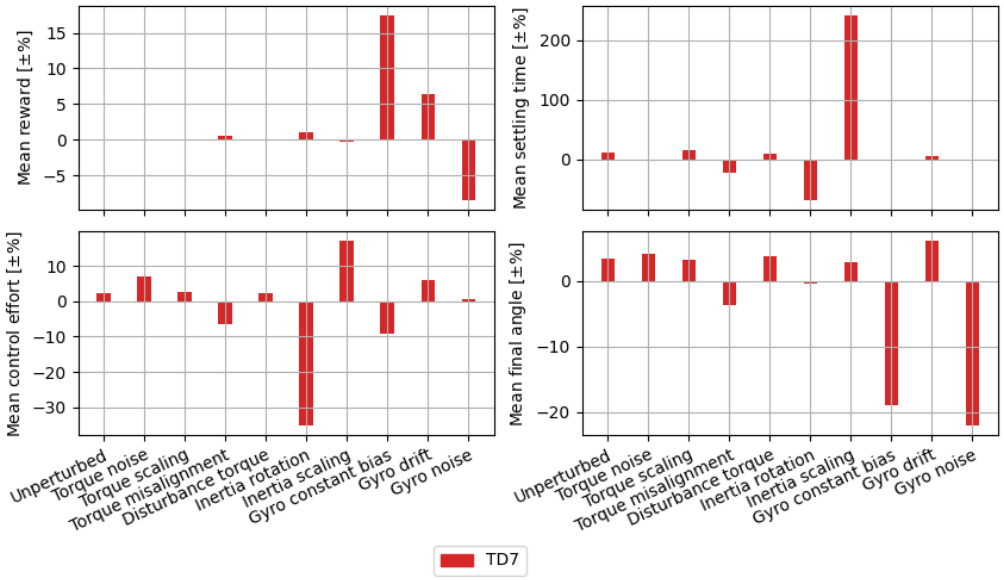


Figure A.23: Performance of agents with domain randomization applied in the form of inertia rotation, with respect to the baseline performance in the flexible case. Bars indicate one standard deviation.

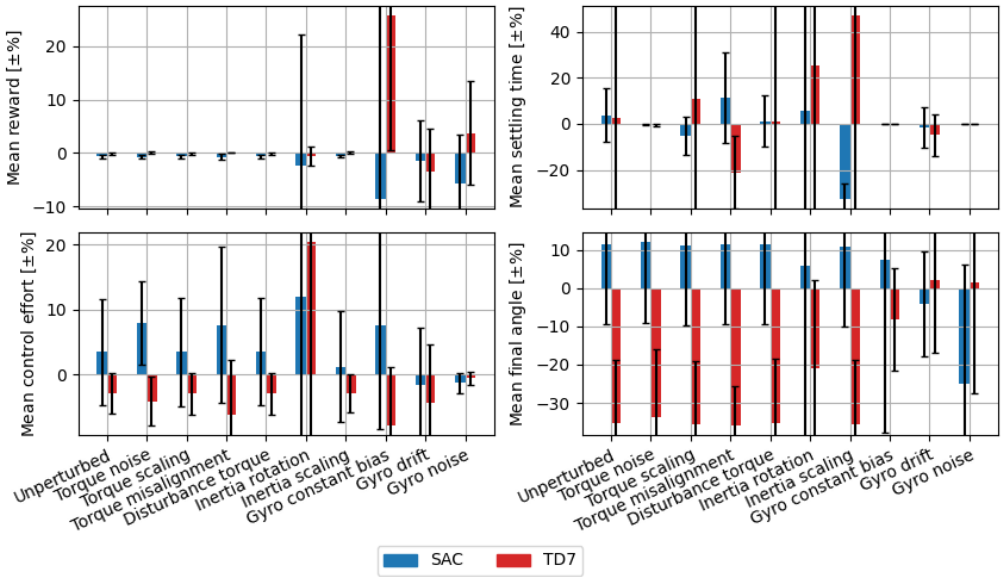


Figure A.24: Performance of agents with domain randomization applied in the form of inertia scaling, with respect to the baseline performance in the flexible case. Bars indicate one standard deviation.

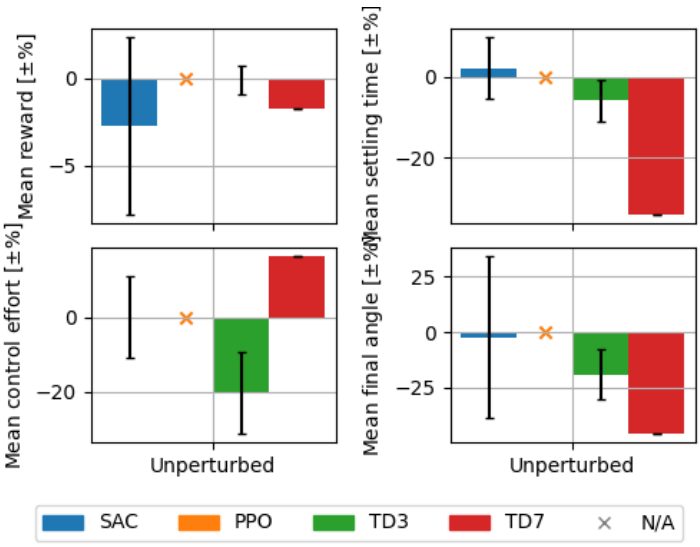


Figure A.25: Performance of agents with the Euler state representation with respect to the baseline performance in the flexible unperturbed case. Bars indicate one standard deviation.

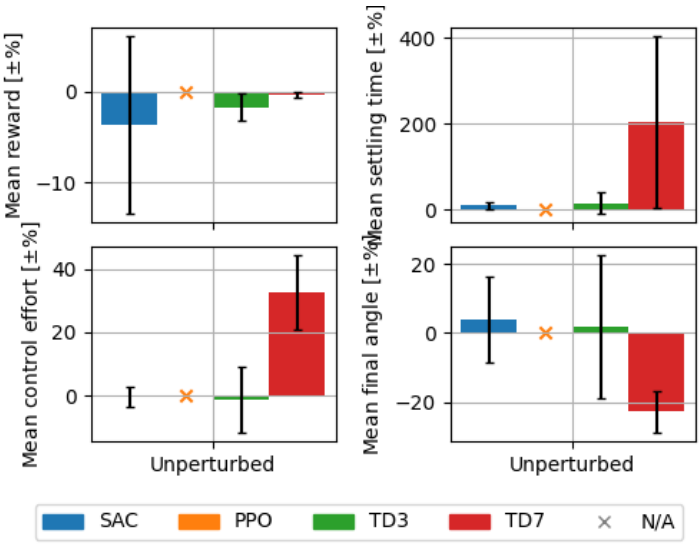


Figure A.26: Performance of agents with the MRP state representation with respect to the baseline performance in the flexible unperturbed case. Bars indicate one standard deviation.

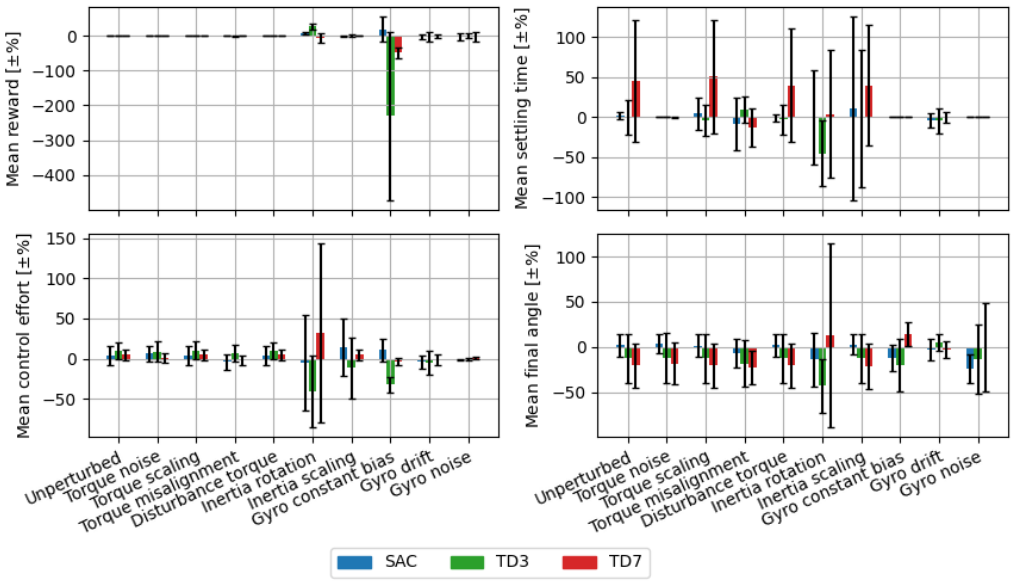


Figure A.27: Performance of agents with domain randomization applied in the form of torque scaling, with respect to the baseline performance in the flexible case. Bars indicate one standard deviation.

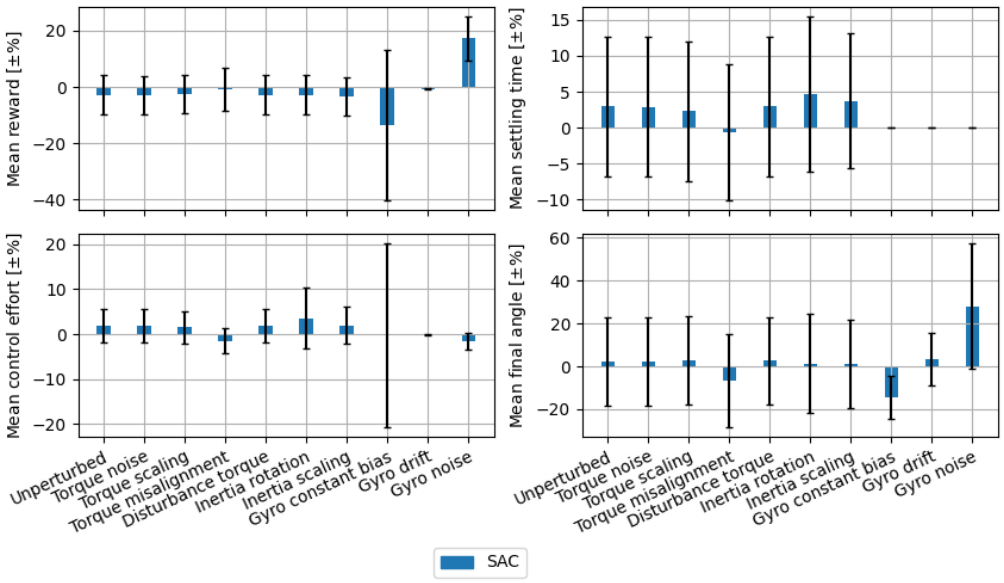


Figure A.28: Performance of agents with domain randomization applied in the form of torque noise, with respect to the baseline performance in the flexible case. Bars indicate one standard deviation.

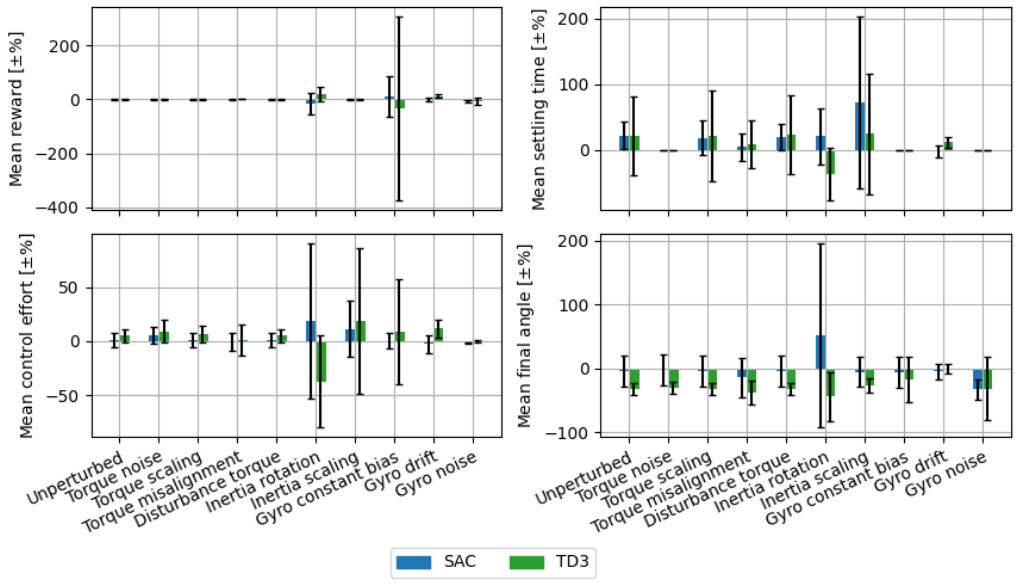


Figure A.29: Performance of agents with domain randomization applied in the form of a disturbance torque, with respect to the baseline performance in the flexible case. Bars indicate one standard deviation.

A.5. Hyperparameter sensitivity - rigid

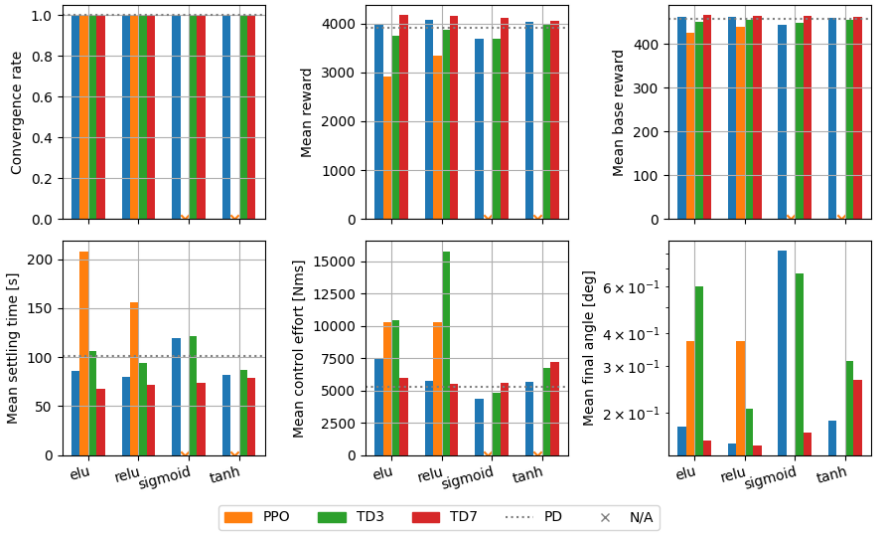


Figure A.30: Performance of the best agents during their training process plotted against the activation function value for all four algorithms in the rigid unperturbed environment. The N/A marker means that no data was available for this case (the sampler did not select this activation function).

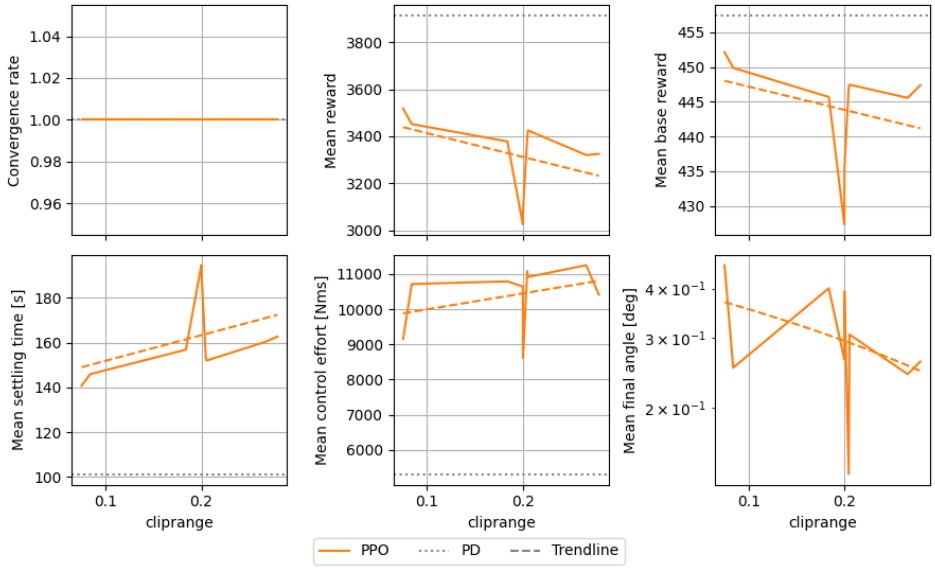


Figure A.31: Performance of the best agents during their training process plotted against the clip range hyperparameter value for PPO in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.18$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

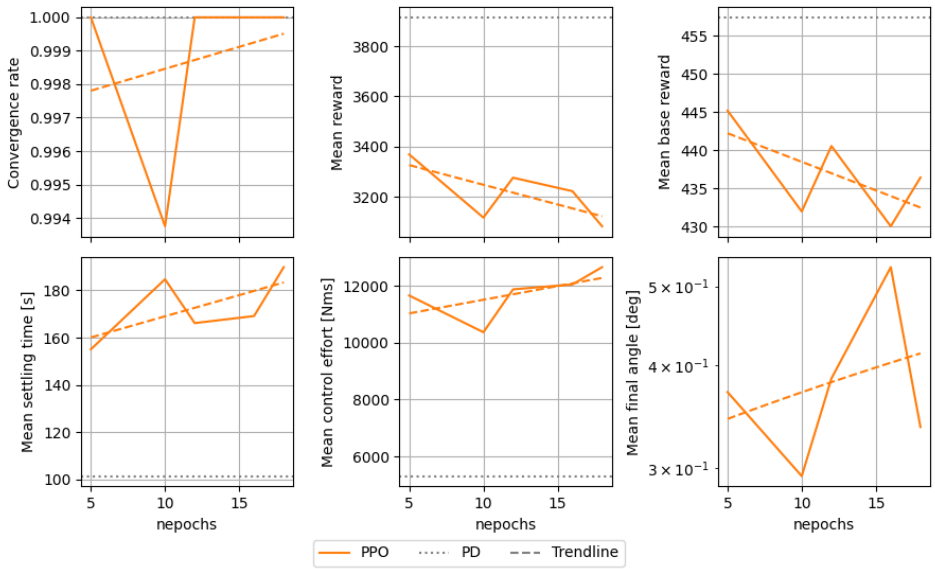


Figure A.32: Performance of the best agents during their training process plotted against the number of epochs hyperparameter value for PPO in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.47$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

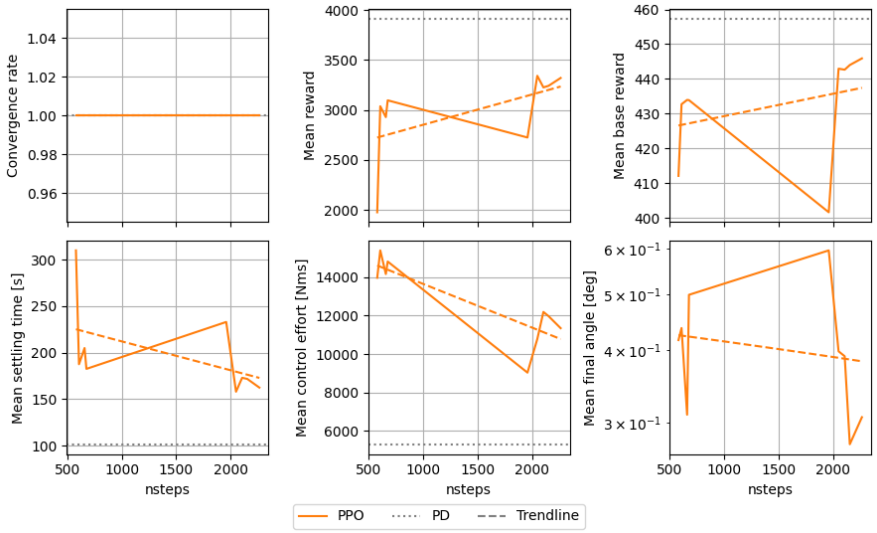


Figure A.33: Performance of the best agents during their training process plotted against the number of steps value for PPO in the rigid unperturbed environment.

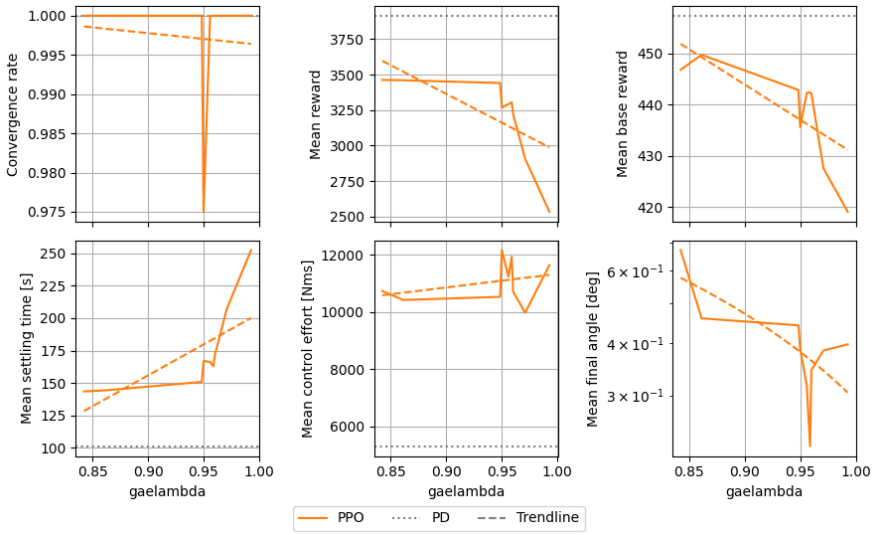


Figure A.34: Performance of the best agents during their training process plotted against the GAE-λ value for PPO in the rigid unperturbed environment.

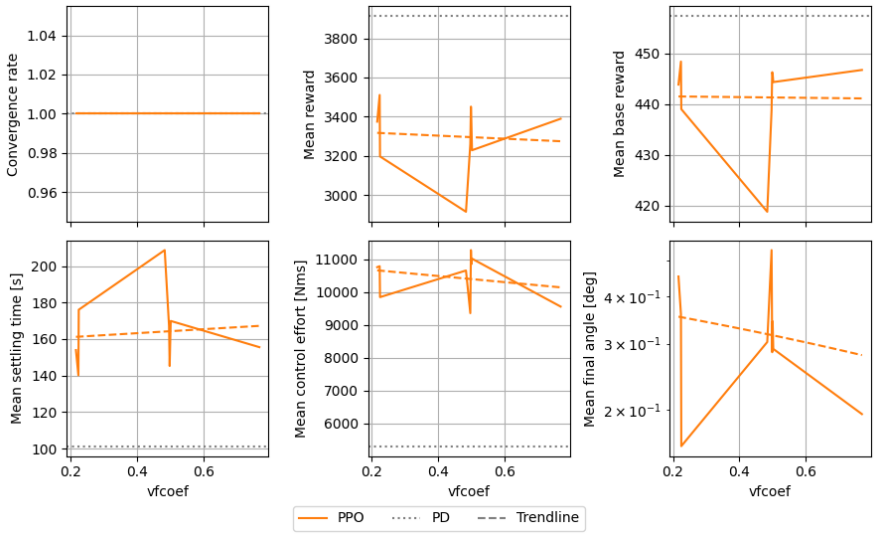


Figure A.35: Performance of the best agents during their training process plotted against the vf-coefficient value for PPO in the rigid unperturbed environment.

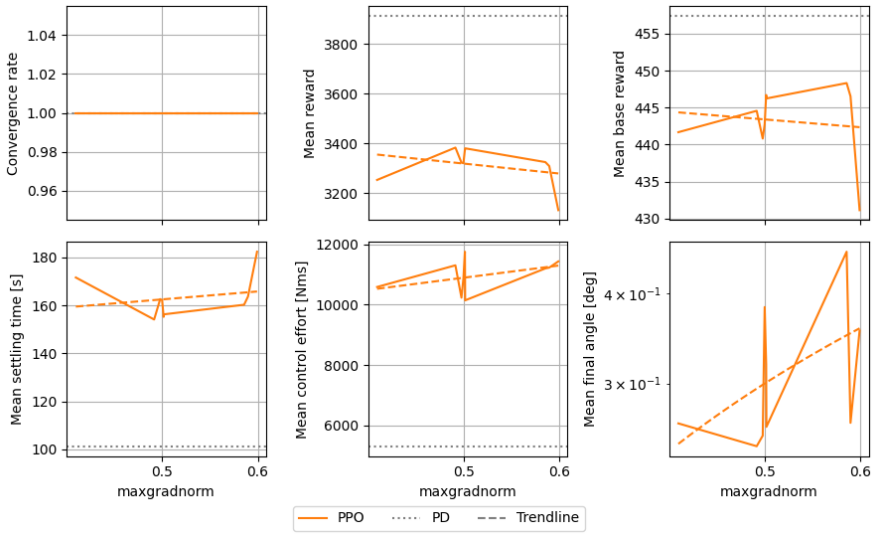


Figure A.36: Performance of the best agents during their training process plotted against the maximum gradient norm value for PPO in the rigid unperturbed environment.

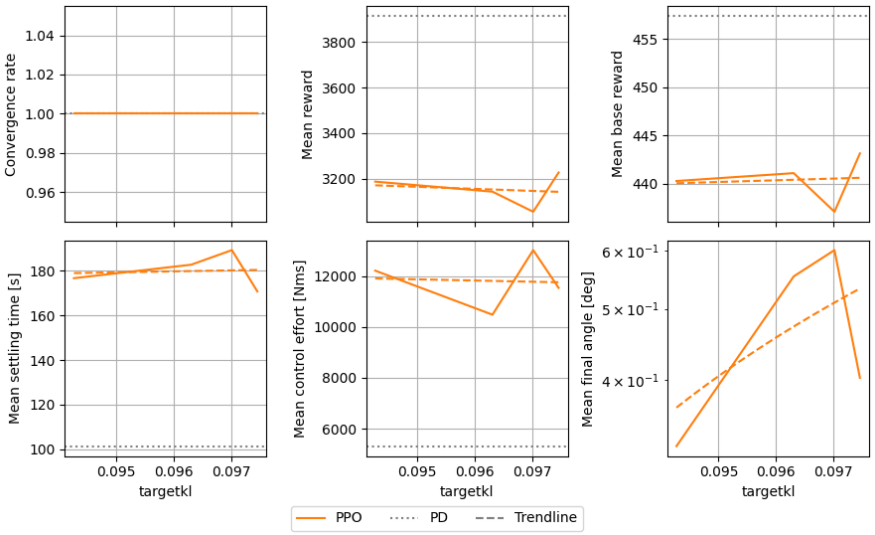


Figure A.37: Performance of the best agents during their training process plotted against the target KL value for PPO in the rigid unperturbed environment.

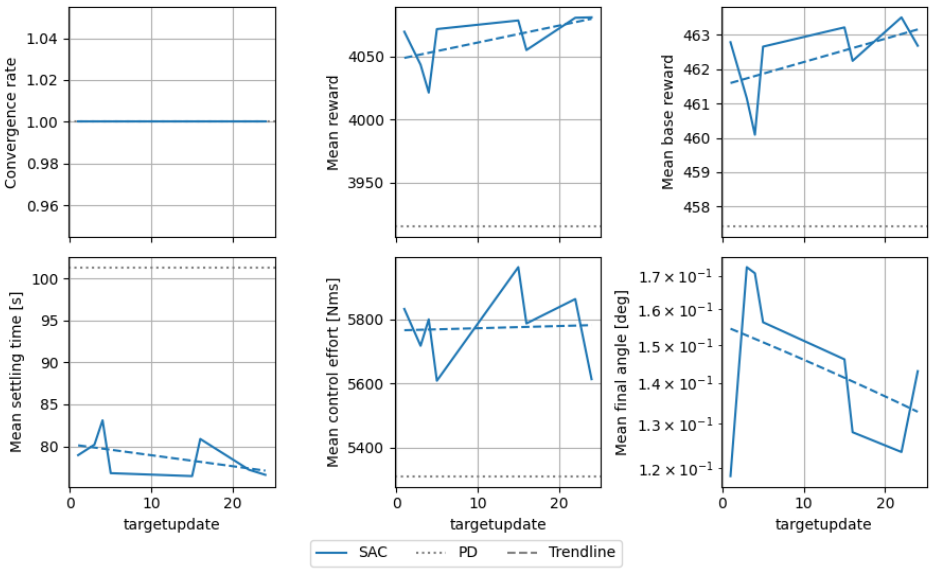


Figure A.38: Performance of the best agents during their training process plotted against the target update interval hyperparameter value for SAC in the rigid unperturbed environment. Of all the trendlines shown (all linear trends), the best fit is $R^2 = 0.33$, which is a bad fit (indicating that the relation is not linear). The trendlines have still been included, to provide a sense of the first-order trend.

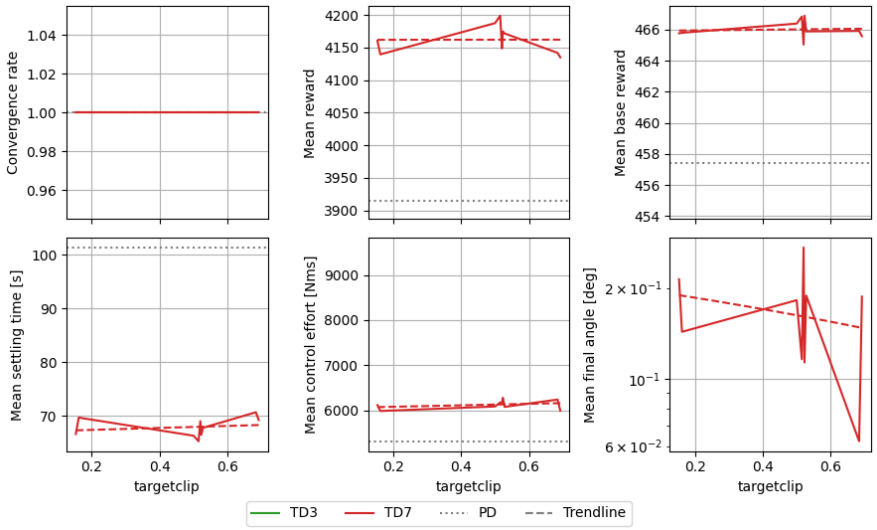


Figure A.39: Performance of the best agents during their training process plotted against the target noise clip value for TD3 and TD7 in the rigid unperturbed environment. Note that no TD3 agent converged.

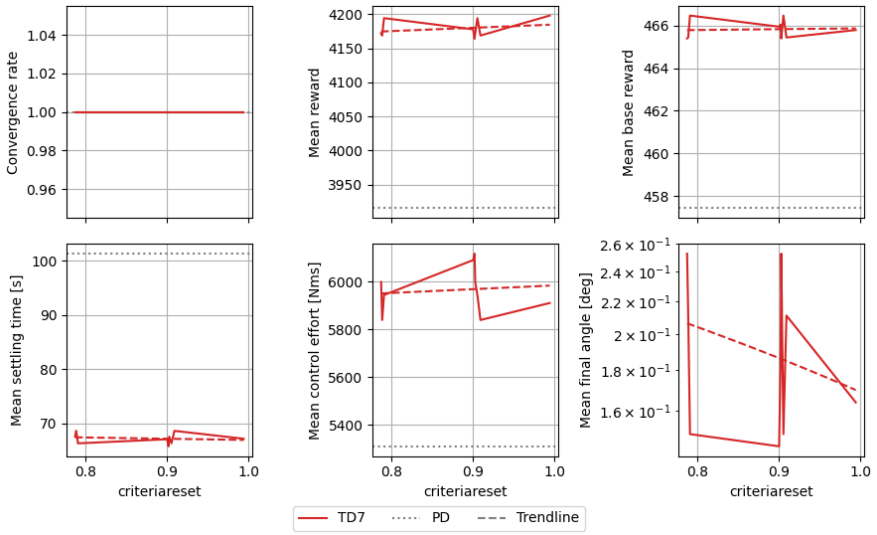


Figure A.40: Performance of the best agents during their training process plotted against the criteria reset value for TD7 in the rigid unperturbed environment.

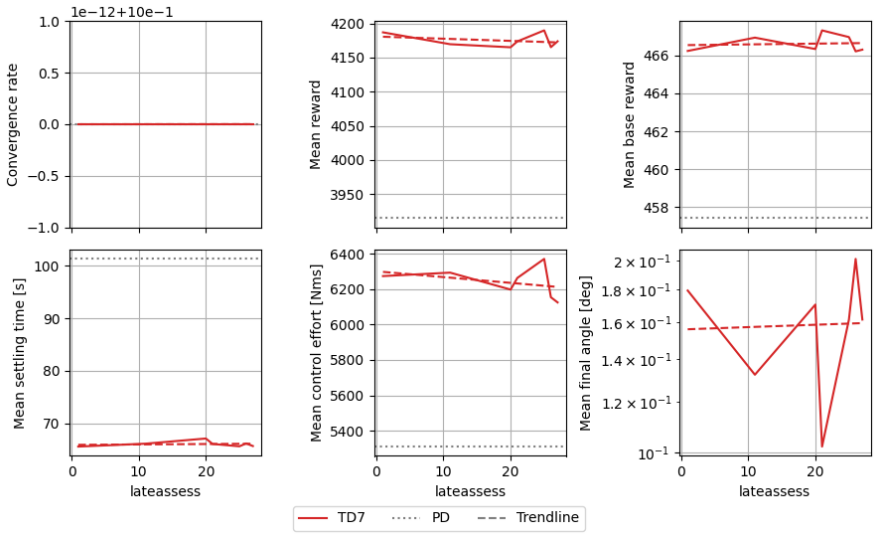


Figure A.41: Performance of the best agents during their training process plotted against the late assessment episodes hyperparameter value for TD7 in the rigid unperturbed environment.

A.6. Hyperparameter sensitivity - flexible

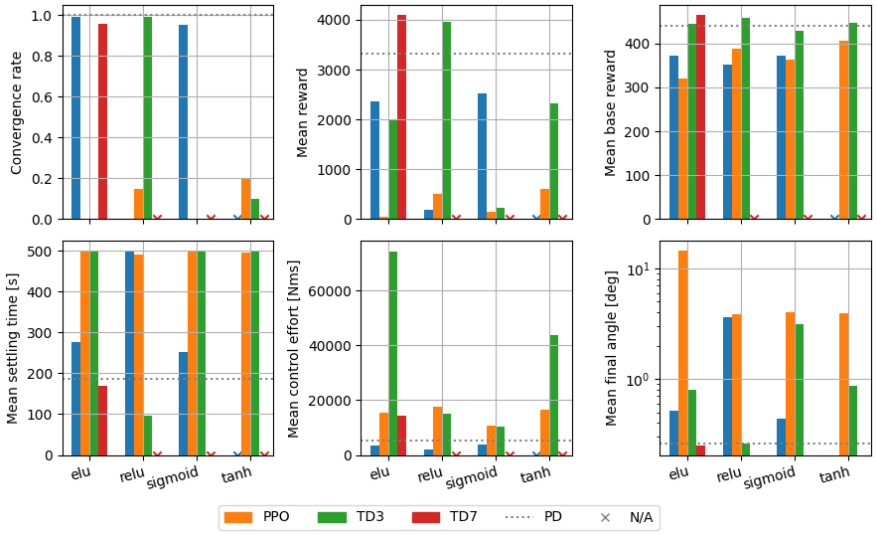


Figure A.42: Performance of the final agents during their training process plotted against the activation function value for all four algorithms in the flexible unperturbed environment. The N/A marker means that no data was available for this case (the sampler did not select this activation function).

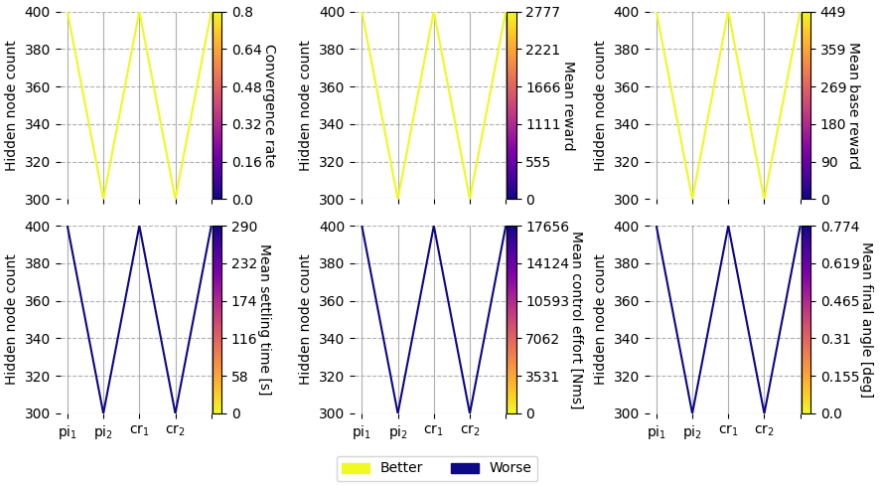


Figure A.43: Average best agent performance metrics for different combinations of node count hyperparameter values for PPO, for the flexible environment. Note that only one agent converged.

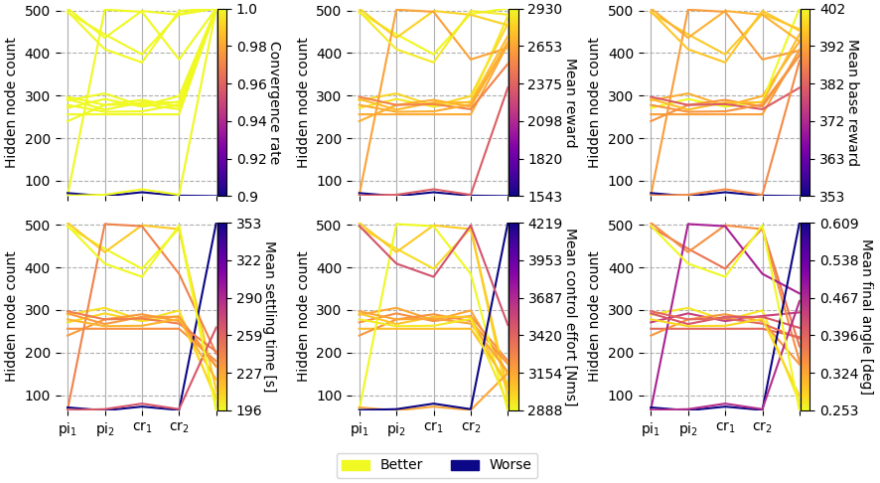


Figure A.44: Average best agent performance metrics for different combinations of node count hyperparameter values for SAC, for the flexible environment.

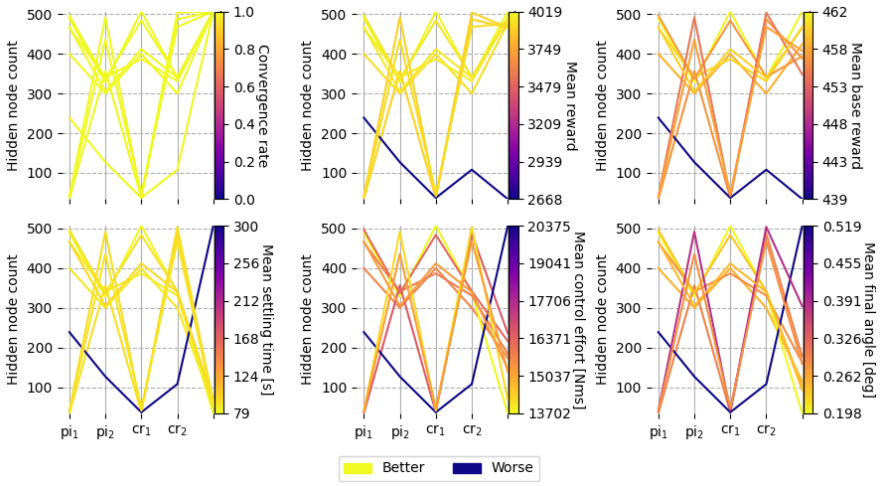


Figure A.45: Average best agent performance metrics for different combinations of node count hyperparameter values for TD3, for the flexible environment.

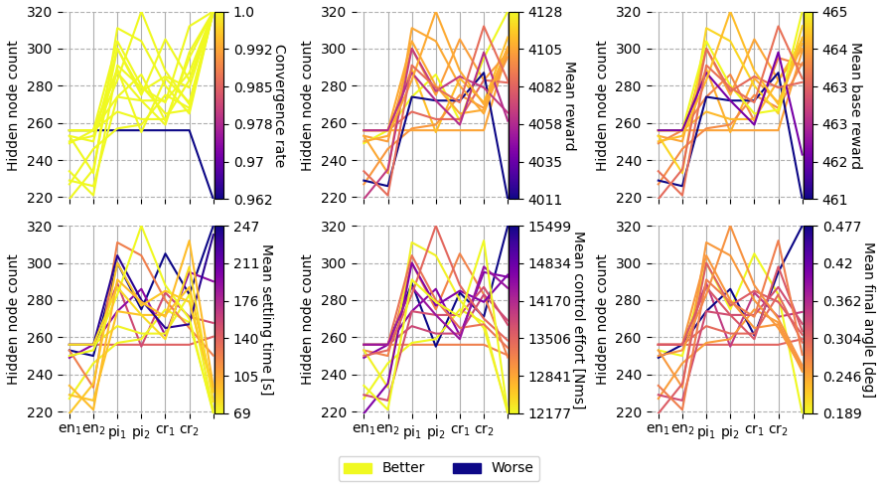


Figure A.46: Average best agent performance metrics for different combinations of node count hyperparameter values for TD7, for the flexible environment.

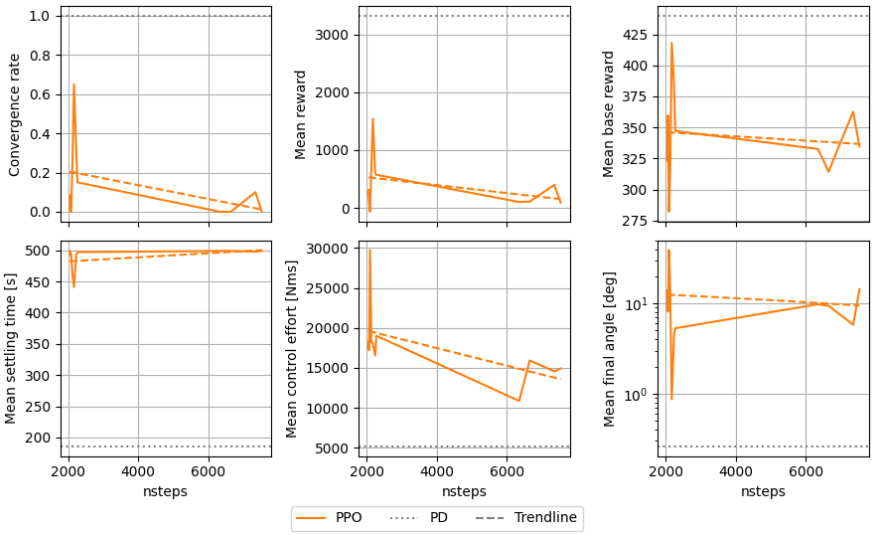


Figure A.47: Performance of the final agents during their training process plotted against the number of steps value for PPO in the flexible unperturbed environment.

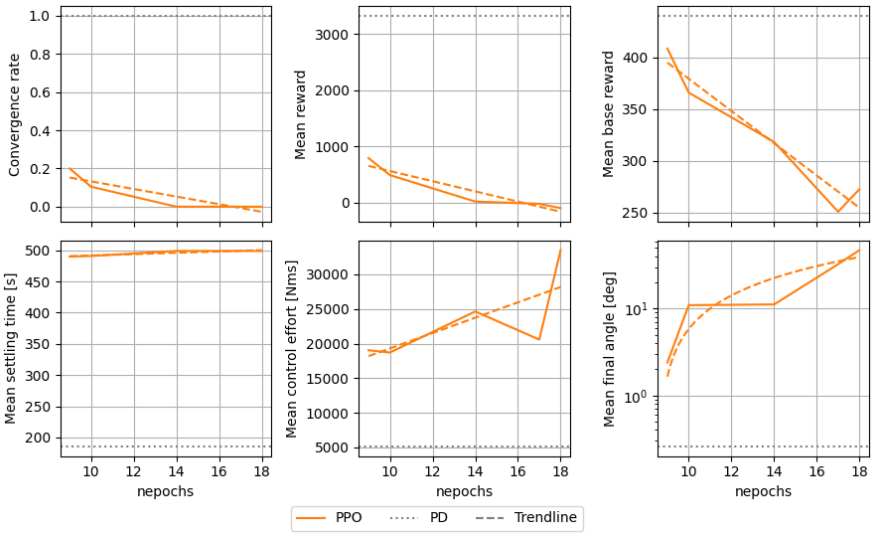


Figure A.48: Performance of the final agents during their training process plotted against the number of epochs value for PPO in the flexible unperturbed environment.

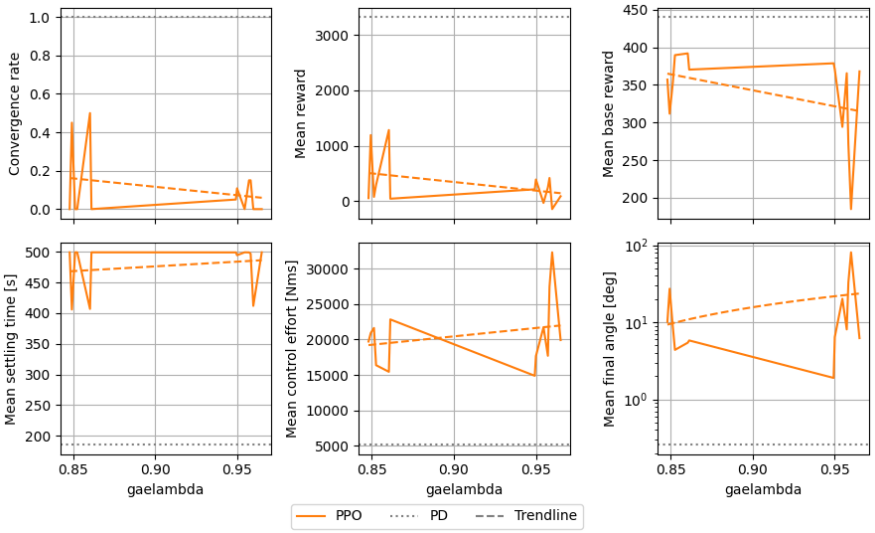


Figure A.49: Performance of the final agents during their training process plotted against the GAE-λ value for PPO in the flexible unperturbed environment.

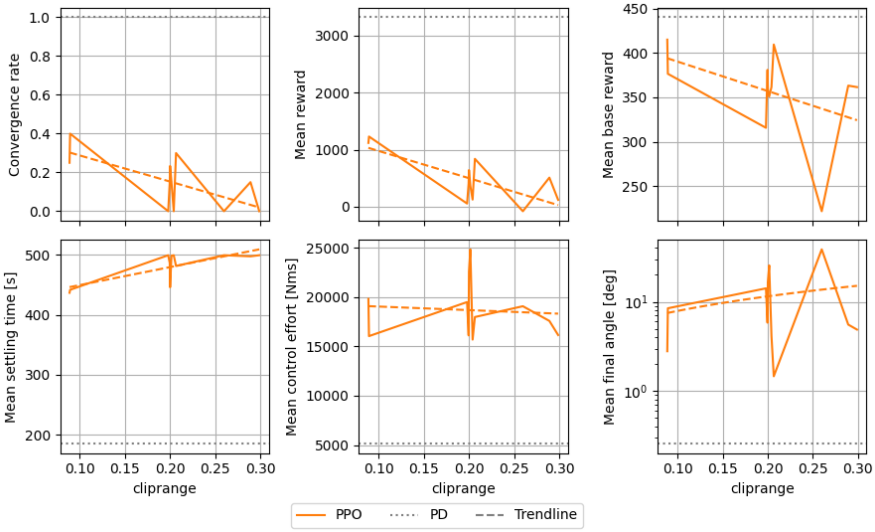


Figure A.50: Performance of the final agents during their training process plotted against the clip range value for PPO in the flexible unperturbed environment.

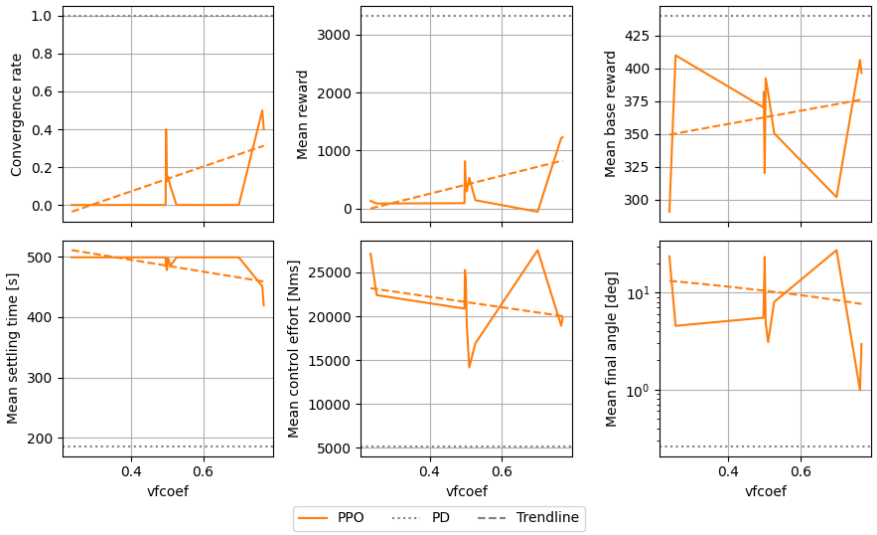


Figure A.51: Performance of the final agents during their training process plotted against the vf-coefficient value for PPO in the flexible unperturbed environment.

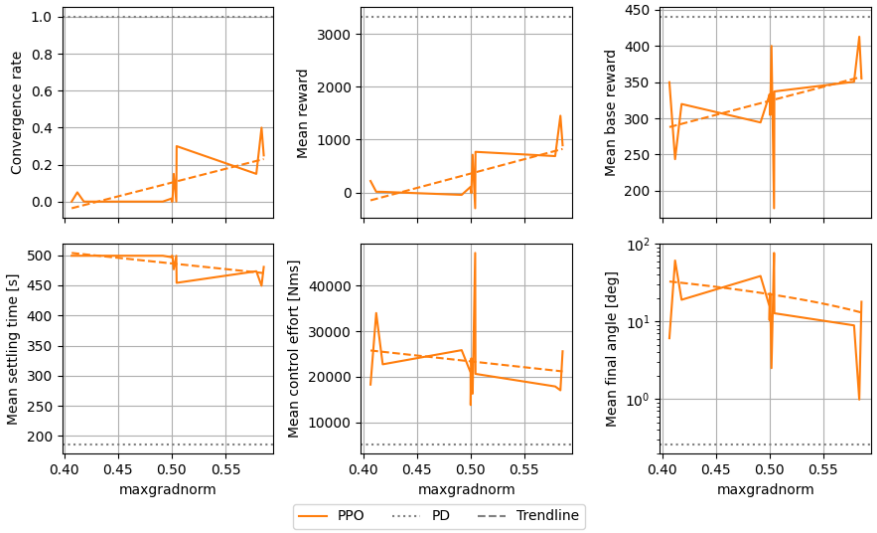


Figure A.52: Performance of the final agents during their training process plotted against the max gradient norm value for PPO in the flexible unperturbed environment.

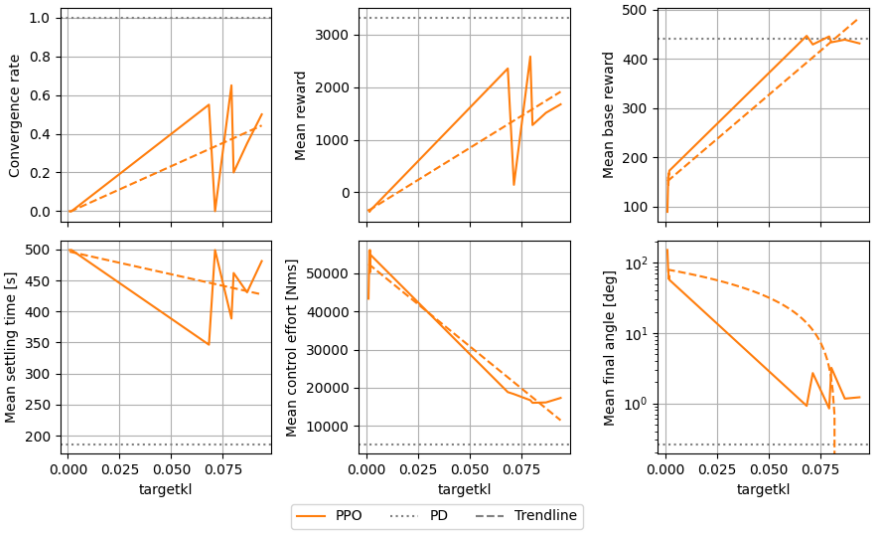


Figure A.53: Performance of the final agents during their training process plotted against the target KL value for PPO in the flexible unperturbed environment.

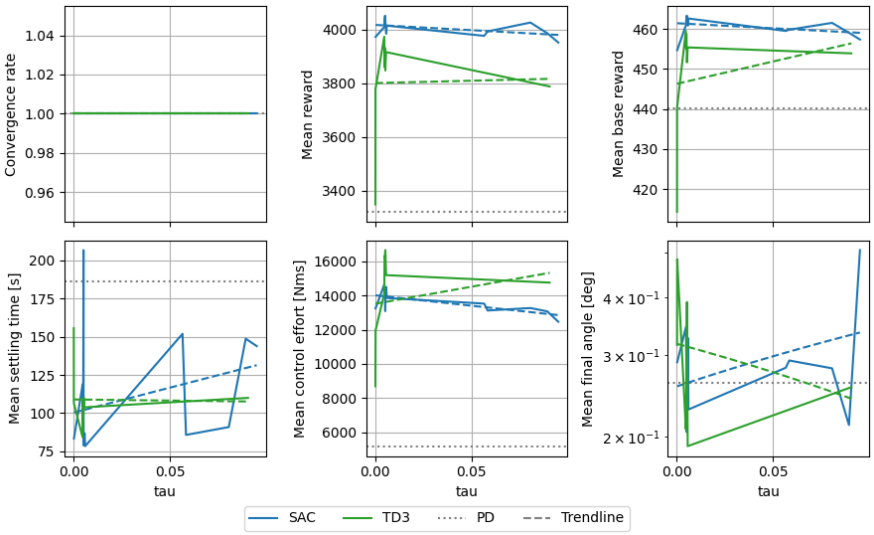


Figure A.54: Performance of the best agents during their training process plotted against the τ value for TD3 and SAC in the flexible unperturbed environment.

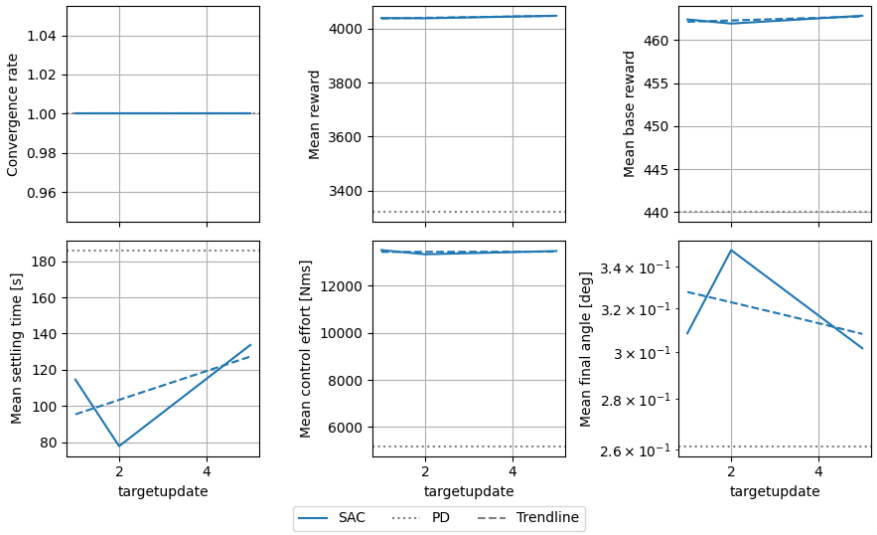


Figure A.55: Performance of the best agents during their training process plotted against the target update hyperparameter value for SAC in the flexible unperturbed environment.

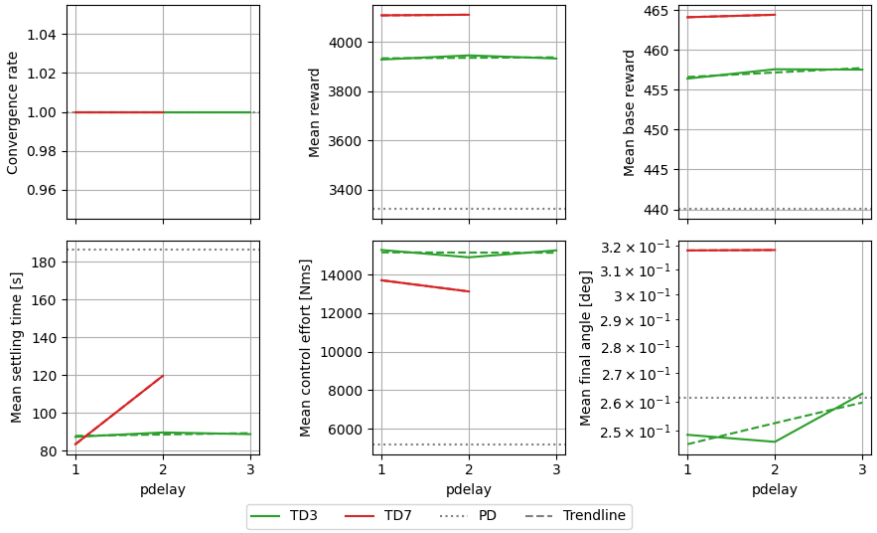


Figure A.56: Performance of the best agents during their training process plotted against the policy delay value for TD3 and TD7 in the flexible unperturbed environment.

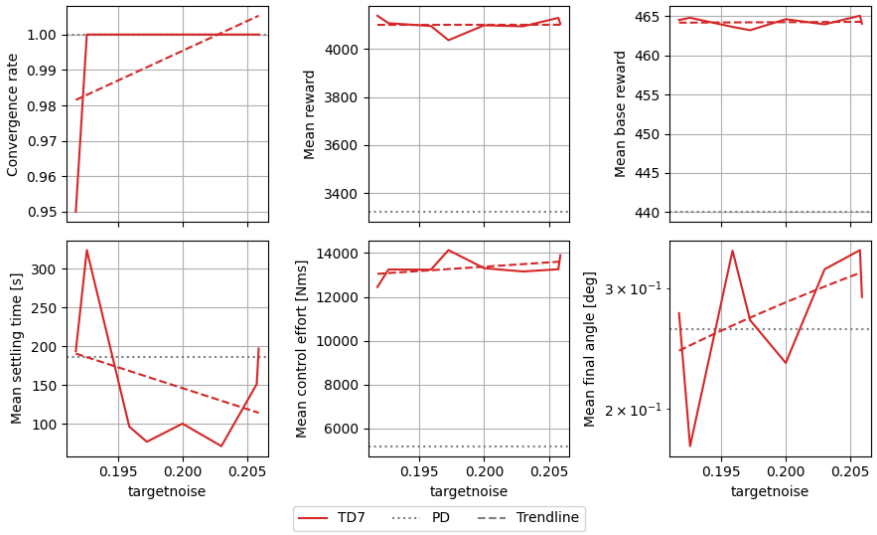


Figure A.57: Performance of the best agents during their training process plotted against the target noise standard deviation value for TD3 and TD7 in the flexible unperturbed environment.

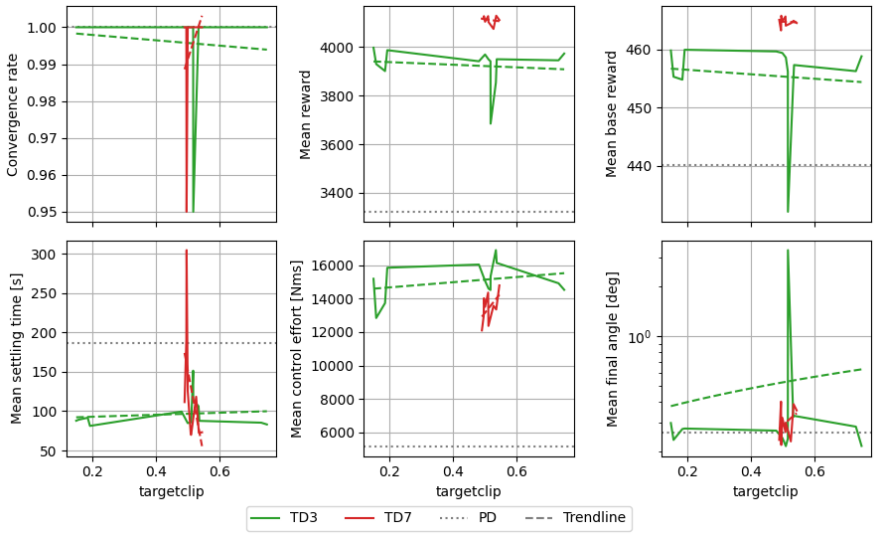


Figure A.58: Performance of the best agents during their training process plotted against the target noise clip value for TD3 and TD7 in the flexible unperturbed environment.

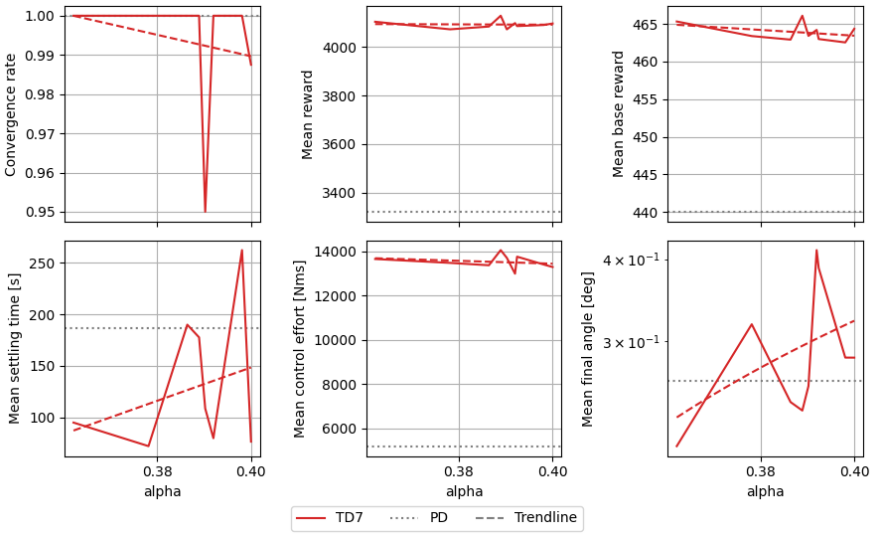


Figure A.59: Performance of the best agents during their training process plotted against the α value for TD7 in the flexible unperturbed environment.

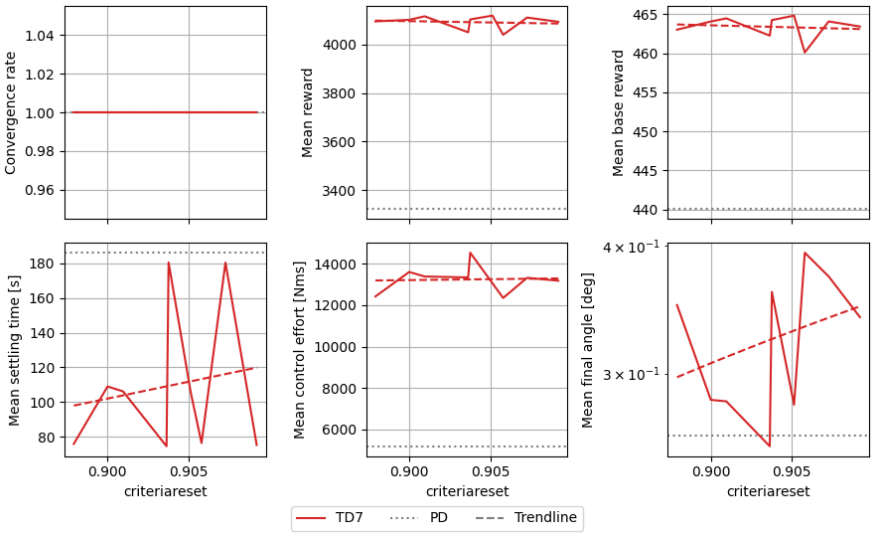


Figure A.60: Performance of the best agents during their training process plotted against the criteria reset value for TD7 in the flexible unperturbed environment.

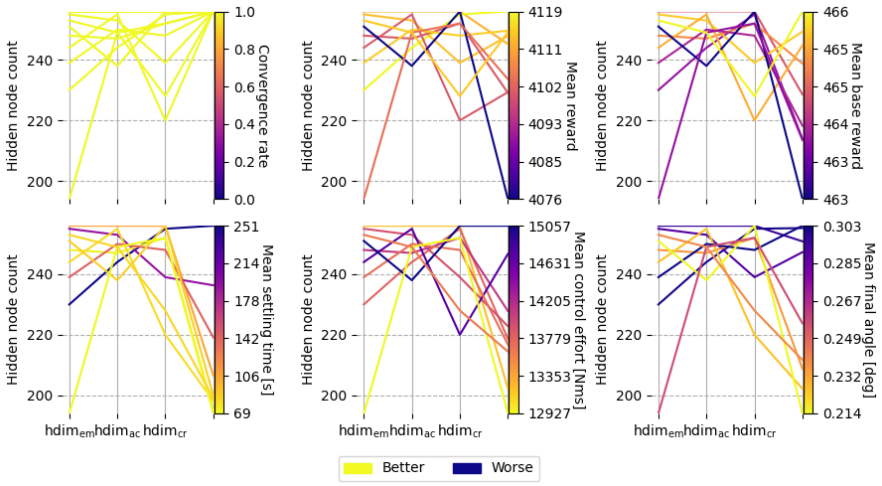


Figure A.61: Performance of the best agents during their training process plotted against the hidden layer dimension size for TD7 in the flexible unperturbed environment.

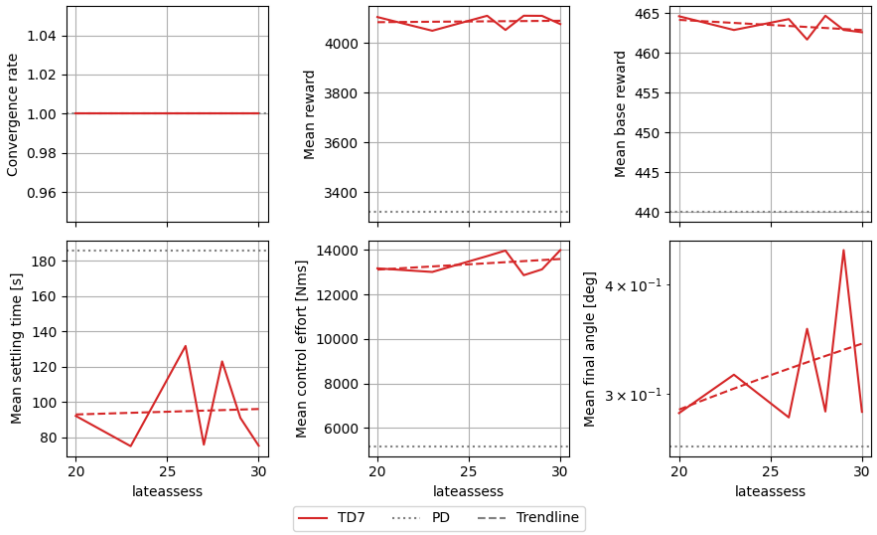


Figure A.62: Performance of the best agents during their training process plotted against the late assessment episodes hyperparameter value for TD7 in the flexible unperturbed environment.

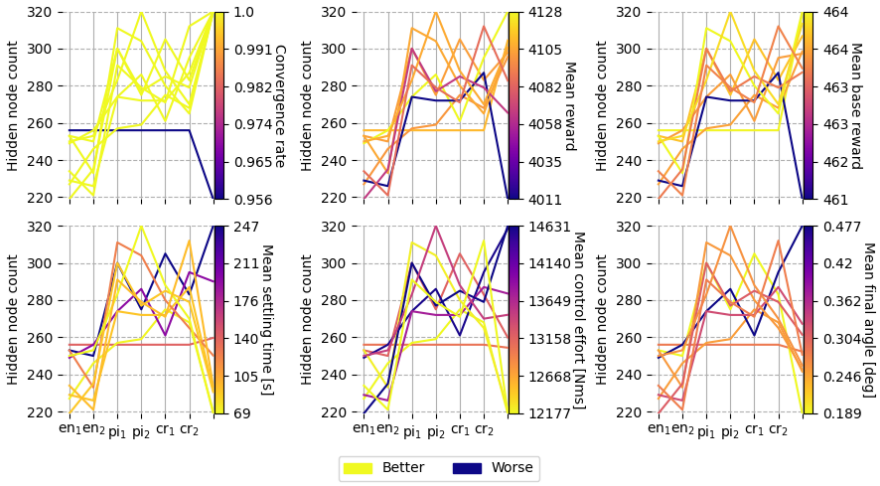


Figure A.63: Performance of the best agents during their training process plotted against the hidden layer node count values (including embedding layers) for TD7 in the flexible unperturbed environment.

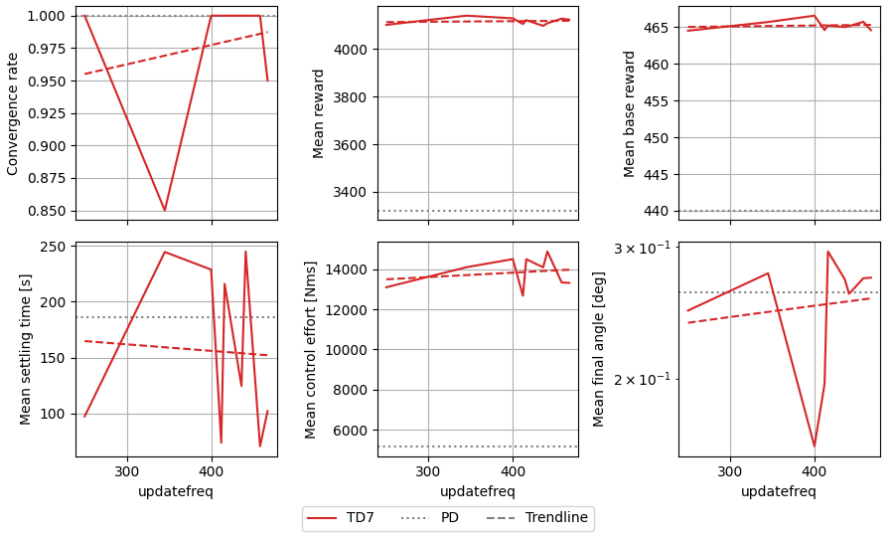


Figure A.64: Performance of the best agents during their training process plotted against the update frequency hyperparameter value for TD7 in the flexible unperturbed environment.