Deep reinforcement learning for spacecraft attitude tracking manoeuvres

Jan Luca Loettgen ⋆†*, Matteo Ceriotti*⋆*, Gerardo Aragon-Camarasa*⋆*, and Kevin Worrall*⋆
University of Glasgow, Glasgow, United Kingdom, G12 8QQ
j.loettgen.1@reserach.gla.ac.uk · matteo.ceriotti@glasgow.ac.uk ·
gerardo.aragoncamarasa@glasgow.ac.uk · kevin.worrall@glasgow.ac.uk
†Corresponding author

Abstract

This paper investigates the application of Deep Reinforcement Learning (DRL) for attitude tracking manoeuvres of a 6U-CubeSat. The desired reference trajectories to be tracked are generated using the quaternion SLERP and SQUAD algorithms. The use of Long Short-Term Memory layers to reconstruct temporal angular velocity information that would otherwise be unobservable by the controller is investigated. It is demonstrated that DRL-based controllers that use a direction cosine matrix-based attitude parameterisation perform better than controllers that use a quaternion attitude parameterisation. The performance of the DRL controller is compared against a tuned quaternion rate feedback controller in Monte Carlo simulations.

1. Introduction

Attitude control is one of the principal abilities required by modern spacecraft. Traditional attitude control strategies use a combination of sensors and state estimators to infer orientation and rotational information. Feedback controllers such as Quaternion Rate Feedback (QRF)[25,26] or sliding mode[2,22] controllers are then used to control actuators that drive the spacecraft to commanded orientations. A new area of research has considered augmenting or replacing the existing attitude control pipeline with Deep Reinforcement Learning (DRL)-based controllers. DRL is a feedback control strategy where a decision-making agent learns to achieve control objectives by interacting with the problem. DRL does not require a model of the problem, only the ability to interact with the problem. DRL has been shown to outperform humans in games, including Atari video games,[12,16] Chess,[20] Go,[19–21] and Shogi.[20] DRL has also been used for motor control of robots.[10,15,16] While most of this research was on simulated environments, DRL agents have also been deployed onto real robots after training.[9] Another example of DRL agents bridging the simulation-to-real gap is for magnetic control of plasma in the Tokamak fusion reactor.[1]

DRL has been applied in the aerospace domain in planetary landing scenarios,[6] missile guidance,[5] and attitude and altitude control of a spacecraft over unknown characteristic asteroids.[7] In the spacecraft attitude control domain, DRL has been proposed for the detumbling of a spacecraft after sudden changes to spacecraft mass distribution.[24] DRL for end-to-end attitude control during large angle slew manoeuvres has been investigated.[3,4,23] In large-angle slew manoeuvres, the objective is to reorient the spacecraft into a specific constant orientation relative to an inertial reference frame. This paper extends this previous research by looking at DRL for end-to-end attitude control in attitude tracking of unknown trajectories. In attitude-tracking scenarios, the spacecraft desired reference orientation is time-varying, and the attitude controller must continuously command torques to follow the reference trajectory. The increased difficulty of this problem over large angle slews is because the time history of attitude information should be considered to predict the required control torques. The relative simplicity of the attitude tracking problem makes it suitable for a case study for DRL. Specifically, the attitude tracking problem is a sub-problem to more complex problems, including full attitude and orbit control of a spacecraft.

To generate reference trajectories, the quaternion SLERP[17] and SQUAD[17] algorithms are used. The SLERP algorithm provides constant angular velocity trajectories between two orientations. Due to the constant angular velocity, all future orientations on the trajectory can be predicted from only measurements of the current reference orientation and angular velocity. The SQUAD algorithm spherically interpolates between 4 unit quaternion control points similar to cubic Bézier curves. SQUAD trajectories are non-constant angular velocities, and so provide a more difficult tracking trajectory. Due to this non-constant angular velocity, the observational input to the DRL controller can lose the Markov property. The observational input to the controller has the Markov property if it contains equivalent information as the time history of observations. This paper investigates the use of Long Short-Term Memory (LSTM) artificial

neural network layers in the DRL policy to infer the missing temporal information. Specifically, the performance of three DRL controllers is compared for SLERP tracking. The first controller is only provided the current reference orientation as input; the second is also provided the angular velocity, and as such, the observation has the Markov property. The third DRL controller is provided with only the position of the reference but utilises LSTM layers to infer the angular velocity of the reference. For SQUAD trajectories, even when the angular velocity is provided to the controller, the observation still loses the Markov Property. For the SQUAD tracking scenario, the performance of three controllers is compared. The first is only provided the reference orientation; the second is also provided the angular velocity. The third is provided the orientation and angular velocity of the reference while also utilising LSTM layers to infer information about the non-constant angular velocity. For both the SLERP and SQUAD tracking scenarios, the performance of the controller is compared against a QRF controller.

In Section 2, the relevant quaternion mathematics and kinematics are provided. Section 3 provides an introduction to reinforcement learning. Section 4.1 provides the dynamic model of the spacecraft. Section 4.2 details how the trajectories to be tracked were generated. Section 4.3 formalises the attitude tracking problem as a discrete-time Markov decision process, including the observation provided to the controller, the actions it can take to control the spacecraft, and the reward function used to train the controller. Section 4.4 provides details on the agents trained, including the architecture of the artificial neural networks and hyperparameters. Lastly, Section 5 compares the performance of the trained agents and discusses the effectiveness of the LSTM in recovering the performance of the Markovian controller.

2. Quaternions

This paper uses a scalar-first convention to parameterise unit quaternions. A unit quaternion describes a rotation by an angle ϕ about an axis, specified by the unit vector $\boldsymbol{e} = (e_1, e_2, e_3)$, using four components:

$$\begin{aligned} q_w &= \cos\left(\frac{\phi}{2}\right) \\ q_1 &= e_1 \sin\left(\frac{\phi}{2}\right) \\ q_2 &= e_2 \sin\left(\frac{\phi}{2}\right) \\ q_3 &= e_3 \sin\left(\frac{\phi}{2}\right) \end{aligned} \tag{1}$$

A unit quaternion $\boldsymbol{q}^{AB}$ describes a frame rotation between two reference frames $\mathcal{F}^A$ and $\mathcal{F}^B$. The quaternion can be converted to a Direction Cosine Matrix (DCM) $\mathbf{C}$:

$$\mathbf{C}^{AB} = \begin{bmatrix} q_w q_w + q_1 q_1 - q_2 q_2 - q_3 q_3 & 2(q_1 q_2 + q_3 q_w) & 2(q_1 q_3 - q_2 q_w) \\ 2(q_1 q_2 - q_3 q_w) & q_w q_w - q_1 q_1 + q_2 q_2 - q_3 q_3 & 2(q_2 q_3 + q_1 q_w) \\ 2(q_1 q_3 + q_2 q_w) & 2(q_2 q_3 - q_1 q_w) & q_w q_w - q_1 q_1 - q_2 q_2 + q_3 q_3 \end{bmatrix} \tag{2}$$

Using Eq. (2), a vector $\boldsymbol{v}_B$ in the $\mathcal{F}^B$ frame can be converted to its corresponding representation in the $\mathcal{F}^A$ frame:

$$\boldsymbol{v}_A = \mathbf{C}^{AB} \boldsymbol{v}_B \tag{3}$$

The angular velocity $\boldsymbol{\omega}_A^{AB} = (\omega_1, \omega_2, \omega_3)$ between the $\mathcal{F}^A$ frame and the frame $\mathcal{F}^B$ as viewed from $\mathcal{F}^A$ can be used to find the rate of change of the quaternion representing the rotation:

$$\dot{\boldsymbol{q}}^{AB} = \frac{1}{2} \boldsymbol{\Xi}(\boldsymbol{q}^{AB}) \boldsymbol{\omega}_A^{AB} \tag{4}$$

$$\boldsymbol{\Xi}(\boldsymbol{q}^{AB}) = \begin{bmatrix} q_w & -q_1 & -q_2 & -q_3 \\ q_1 & q_w & -q_3 & q_2 \\ q_2 & q_3 & q_w & -q_1 \\ q_3 & -q_2 & q_1 & q_w \end{bmatrix} \tag{5}$$

where $\boldsymbol{\omega}_A^{AB}$ is treated as a pure quaternion with 0 in the scalar part. The inverse of Eq. (5) can be used to find the angular velocity from the quaternion rate of change:

$$\boldsymbol{\omega}_A^{AB} = \boldsymbol{\Xi}^T(\boldsymbol{q}^{AB}) \dot{\boldsymbol{q}}^{AB} \tag{6}$$

$$\boldsymbol{\Xi}^T(\boldsymbol{q}^{AB}) = \begin{bmatrix} -q_1 & q_w & q_3 & -q_2 \\ -q_2 & -q_3 & q_w & q_1 \\ -q_3 & q_2 & -q_1 & q_w \end{bmatrix} \tag{7}$$

3. Reinforcement Learning

Reinforcement learning is a framework for solving discrete-time Markov Decision Processes (MDP). MDPs formalise the sequential decision-making of an agent in an environment. At each step in the MDP, the environment provides an observation to the agent, typically a vector of sensor measurements or images. The observation is a representation of the internal state of the environment. Based upon the observation, the agent selects an action, typically actuator commands. The action causes the environment to transition into a new state. The environment then provides a new observation to the agent and a scalar reward. The scalar reward informs the agent how desirable the current internal state of the environment is. Based upon the new observation, the agent selects a new action, and the process repeats until the environment transitions into a terminal state that ends the current episode of interactions between the agent and the environment. The goal of the agent is to select actions that maximise the discounted sum of future rewards:

$$G_t = \sum_{l=t}^{K-1} \gamma^{l-t} r_l \tag{8}$$

where K is the number of steps in the episode horizon, r_t is the scalar reward provided to the agent after step t. $\gamma \in [0, 1)$ is a constant discount factor that causes the agent to value rewards that will be received sooner more than rewards that will be received later.

In reinforcement learning, the agent employs a decision-making policy $\pi(a|o)$ that maps observations o to actions a. DRL utilises Artificial Neural Networks (ANNs) to parameterise the behavioural policy. Specifically, an ANN with parameters θ_π is used to parameterise the behavioural policy $\pi(a|o, \theta_\pi)$, the goal is then to find parameters θ_π that maximise the discounted sum of future rewards. The ANN takes as input the observation and outputs probabilities that are used to generate actions. Initially, the parameters θ_π are random, and the agent attempts to learn parameters that generate actions that maximise the discounted sum of future rewards. To do this, the agent interacts with the environment for a specific number of steps. The observations seen, actions taken, and rewards received are recorded and used to update the parameters θ_π towards parameters that provide a better behavioural policy using gradient descent. The exact loss function used depends on the DRL algorithm being employed, in this case, the clipped Proximal Policy Optimisation (PPO) algorithm.[16] The PPO algorithm was selected as it provides competitive performance on continuous action space control problems and also has a recurrent LSTM implementation available in the software package Stable-Baselines3.[13] The PPO algorithm is an actor-critic algorithm in the family of policy gradient algorithms. The PPO algorithm runs the current agent for $M \leq K$ time-steps in N copies of the environment while recording the taken actions, seen observations, and received rewards. A loss function is used to update the behavioural policy with the recorded data. PPO clips the updates to the parameters θ_π to be within a specified range. This is done to limit destructive updates to the policy. For a more detailed explanation of the PPO algorithm, the reader is referred to Ref. 16.

The observations provided to the agent are usually sensor measurements, provided as either a vector or a matrix in the case of a camera sensor. If the observation does not have the Markov property (provides equivalent information as full observation and action history), then the MDP is considered to be partially observable. In a Partially-Observable Markov Decision Process (POMDP), the observation provided to the agent is a function of the true internal state (s_t) of the environment ($o_t = H(s_t)$). An example of this is when the measurements of the sensors that generate the observation are subject to noise. The observation vector loses the Markov property if some information can only be observed from a time series of observations. A lack of ability to remember previous states can prevent the agent from being able to learn a successful behavioural policy. In this case, it is common to use LSTM layers[8] in the ANN. LSTM layers are recurrent ANNs layers that allow the agent to propagate information across time steps. Each LSTM layer is made up of a specified number of cells, and as such can only store limited information. When combined with DRL, then the agent must learn what information to store inside the LSTM cells while simultaneously learning the behavioural policy.

4. Methodology

4.1 Spacecraft Model

Three reference frames are considered in this paper: the spacecraft body frame $\mathcal{F}^B$, in which control torques are applied and the dynamics are calculated; the target reference frame $\mathcal{F}^R$, which defines the desired orientation of the spacecraft; and the inertial reference frame $\mathcal{F}^N$, with respect to which the spacecraft tracks its own orientation and relative to which the desired reference orientation is defined. Subscripts indicate the reference frame in which a quantity is measured and two consecutive superscripts are used to indicate a rotation of one reference frame relative to another. A quantity $\boldsymbol{L}_B$ is a vector measured in the body reference frame and a quantity $\boldsymbol{\omega}_B^{BN}$ is a vector related to the rotation of the body reference frame relative to the inertial reference frame when measured in the body reference frame. The 6U-CubeSat is modelled as a rigid uniform-density cuboid with side lengths of 30 cm, 20 cm, and 10 cm and a mass of 6 kg. A model of the spacecraft and its body axes is shown in Fig. 1.

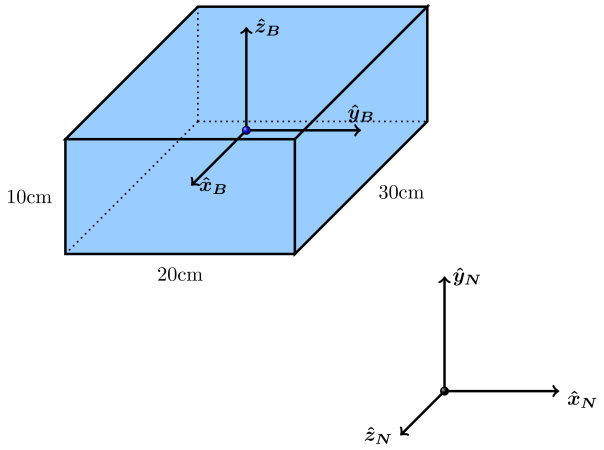

Figure 1: Body frame, inertial reference frame, and dimensions of 6U-CubeSat

The inertia tensor of the spacecraft in body axes is calculated from the side lengths and mass of the cuboid:

$$\boldsymbol{J}_B = \frac{m}{12}\begin{bmatrix} w^2+h^2 & 0 & 0 \\ 0 & l^2+h^2 & 0 \\ 0 & 0 & l^2+w^2 \end{bmatrix} = \begin{bmatrix} 0.025 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.065 \end{bmatrix} \text{kgm}^2 \tag{9}$$

The rotational dynamics of the rigid body are governed by Euler's equation of rotational motion[14] in the body frame:

$$\boldsymbol{J}_B\dot{\boldsymbol{\omega}}_B^{BN} = \boldsymbol{L}_B - \boldsymbol{\omega}_B^{BN} \times (\boldsymbol{J}_B\boldsymbol{\omega}_B^{BN}) \tag{10}$$

where $\boldsymbol{\omega}_B^{BN}$ is the angular velocity of the body frame relative to the inertial frame and $\boldsymbol{L}_B$ are the applied external control torques. The attitude parameterisation considered is unit quaternions. The unit quaternion $\boldsymbol{q}^{BN}$ parameterises the rotation required to move from the inertial reference frame to the body reference frame. The rate of change of the quaternion attitude representation is found from the angular velocity and current attitude quaternion using Eq. (4). Eqs. (4) and (10) are sufficient to fully define the orientation of the spacecraft. To simulate spacecraft rotation, these equations can be numerically integrated; in this case, the Basilisk astrodynamics simulator[1] was used to provide fast simulations. The dynamics are integrated using the Runge-Kutta 4$^{\text{th}}$ order method with a time step of 0.005 s; this simulation time step was found to provide stable simulations for rotations of less than 1 rad/s and be sufficiently fast to support thousands of attitude tracking simulations.

4.2 Trajectory Generation

4.2.1 Quaternion SLERP

To randomly generate time-series quaternion reference trajectories, the quaternion Spherical Linear intERPolation (SLERP) algorithm is used.[17] The SLERP algorithm interpolates linearly between two endpoint quaternions $\boldsymbol{q}_1$, $\boldsymbol{q}_2$,

[1]Note, the Basilisk astrodynamics simulator actually uses a modified Rodriguez parameter attitude parameterisation to simulate the dynamics

using an interpolation parameter u, that represents the normalised manoeuvre time $u = t/T$, where t is the current time in the simulation and T is the total manoeuvre time (100 s). The resulting quaternion $\boldsymbol{q}^{RN}$ describes the reference orientation relative to the $\mathcal{F}^N$ frame:

$$\boldsymbol{q}^{RN}(u) = \text{SLERP}(\boldsymbol{q}_1, \boldsymbol{q}_2, u) = \boldsymbol{q}_1(\boldsymbol{q}_1^* \boldsymbol{q}_2)^u \tag{11}$$

where $\boldsymbol{q}_1^*$ is the conjugate quaternion of $\boldsymbol{q}_1$. Note, consecutive quaternions (e.g. $\boldsymbol{q}_1^* \boldsymbol{q}_2$) such as in Eq. (11) indicates quaternion multiplication. To generate a time series of N quaternions, u is varied linearly from 0 to 1 in N steps. When the quaternion trajectory is used to parameterise rotations, the result is a constant angular velocity rotation, departing from $\boldsymbol{q}_1$ at $u = 0$ and arriving at $\boldsymbol{q}_2$ at $u = 1$, along great-circle arcs. The quaternion rate of change with respect to u associated with the SLERP is found by differentiating Eq. (11):

$$\frac{d\boldsymbol{q}^{RN}}{du} = \boldsymbol{q}^{RN} \log(\boldsymbol{q}^{RN}) \tag{12}$$

Through the chain rule, the time derivative of the quaternion trajectory is found as:

$$\frac{d\boldsymbol{q}^{RN}}{dt} = \frac{1}{T} \boldsymbol{q}^{RN} \log(\boldsymbol{q}^{RN}) \tag{13}$$

which can be converted to angular velocity $\boldsymbol{\omega}_R^{RN}$ using Eq. (6).

4.2.2 Quaternion SQUAD

Another method for generating quaternion trajectories is the SQUAD algorithm,[17] which interpolates between 4 quaternions $\boldsymbol{q}_1$, $\boldsymbol{q}_2$, $\boldsymbol{q}_3$, $\boldsymbol{q}_4$. The SQUAD interpolation between the 4 unit quaternions is similar to cubic Bézier curves but using the SLERP for the individual interpolations instead of linear interpolation:

$$\boldsymbol{q}^{RN}(u) = \text{SQUAD}(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3, \boldsymbol{q}_4, u) = \text{SLERP}(\text{SLERP}(\boldsymbol{q}_1, \boldsymbol{q}_2, u),\ \text{SLERP}(\boldsymbol{q}_3, \boldsymbol{q}_4, u),\ 2u(1-u)) \tag{14}$$

The resulting trajectory has $\boldsymbol{q}^{RN}$ departing from $\boldsymbol{q}_1$ at $u = 0$ towards $\boldsymbol{q}_2$ and arriving at $\boldsymbol{q}_4$ at $u = 1$ from $\boldsymbol{q}_3$. when used to generate smooth rotation trajectories, the associated angular velocity is not constant. The differentiation of Eq. (14) with respect to u is non-trivial due to the fact that the endpoint quaternions in the SLERP become functions of time as opposed to constants. The SQUAD derivative does not, to the authors' knowledge, have a closed-form general solution in terms of the endpoint quaternions and the interpolation factor only. Therefore, the SQUAD is differentiated numerically:

$$\frac{d\boldsymbol{q}^{RN}}{du} \approx \frac{\boldsymbol{q}^{RN}(u + \Delta u) - \boldsymbol{q}^{RN}(u)}{\Delta u} \tag{15}$$

where $\Delta u = \frac{\Delta t}{T}$ is the interpolation step size, and ΔT is the control time step (0.2 s). The angular velocity associated with the SQUAD is found via Eq. (6). Due to the numerical Euler forward differentiation, this generates only $N - 1$ points in the angular velocity trajectory. However, measurements at $t = T$ are not required as the simulation ends here and there are no further actions to take. Hence the angular velocity for the final point is not necessary. As with the SLERP, the endpoint quaternions are randomly generated.[18]

As the quaternion trajectory generated by the SLERP algorithm is a constant angular velocity trajectory, the future quaternions on the trajectory are defined by the angular velocity and the starting point only. As such, a vector containing the angular velocity and current quaternion will have the Markov property (provide equivalent knowledge as knowing the entire trajectory). The angular velocities associated with the SQUAD trajectory are not constant, and as such, measurements of the angular velocity and current quaternion provide insufficient information to predict the future quaternions on the trajectory.

4.3 Markov Decision Process

4.3.1 Observation and Action Space

To apply DRL to the spacecraft attitude control problem, the problem must be converted into a discrete-time POMDP. This requires defining: the observation provided to the agent at every step; the actions the agent can take; how the internal state of the environment transitions between states; and the function that maps the internal state of the environment to rewards that will be provided to the agent. The internal state of the environment tracks the attitude information of the reference and the spacecraft and is propagated forward in time by simulating the dynamics presented in equations

Eqs. (4) and (10). At the start of every time step, the agent receives an observation vector $\boldsymbol{o_t}$ based upon which it selects an action $\boldsymbol{a_t}$. In the observation vector, flattened direction cosine matrices are used to parameterise rotations in place of quaternions, as it was found that the two-to-one attitude parameterisation of quaternions degraded performance. Specifically, the DRL agent only learned to target an error quaternion $\boldsymbol{q}^{BR}$ with a scalar component of $+1$ or -1 but not both, which forced the agent to take the long rotation 50% of the time. The observation vector:

$$\boldsymbol{o_t} = \begin{bmatrix} \mathbf{C}^{BN} & \mathbf{C}^{RN} & \mathbf{C}^{BR} & \boldsymbol{\omega}_B^{BN} & \boldsymbol{\omega}_R^{RN} & \boldsymbol{\omega}_B^{BR} \end{bmatrix} \tag{16}$$

contains information as to the orientation and angular velocity of the spacecraft and the reference, as well as the relative orientation and angular velocity between the spacecraft and reference. The actions taken by the agent are the control torques to be applied to the spacecraft in the body axes frame. The controller receives an observation and takes an action every 0.2 seconds; between control time steps, the applied control torques are held constant as a zero-order hold. The control torques are subject to the constraint that the torque about any body axis, i should not exceed 0.005 Nm in magnitude:

$$\boldsymbol{L}_{B,i}(t) = \begin{cases} 0.005, & \text{if } \boldsymbol{a}_{t,i} \geq 0.005 \text{ Nm} \\ -0.005, & \text{if } \boldsymbol{a}_{t,i} \leq -0.005 \text{ Nm} \\ \boldsymbol{a}_{t,i}, & \text{otherwise} \end{cases} \tag{17}$$

4.3.2 Reward function

After the agent selects an action, the control torques are applied to the spacecraft, whose dynamics are numerically simulated for 0.2 s. The agent then receives a new observation and a scalar reward. The reward function calculates the reward based upon the current rotation angle $\phi(t) = 2\arccos(q_w)$ between the reference orientation and the spacecraft orientation:

$$r_{t,1} = \log_2\left(\frac{|\phi(t-\Delta t)|}{|\phi(t)|}\right) \tag{18}$$

where Δt is the control time step of 0.2 s. This reward function provides a reward equal to 1 every time the pointing error is halved. This reward function was selected as it encourages the agent to reduce the pointing error even if the size of the pointing error is small. The reward function is undefined for both $\phi(t) = 0$ and $\phi(t - \Delta t) = 0$. To avoid this issue, the pointing error used for the computation is clipped to be greater than 10^{-6} rad. In practice, this is not an issue as the agent does not achieve this level of tracking accuracy and sensors on the spacecraft would have difficulty measuring the spacecraft's attitude to this precision.

Additionally, a hard angular velocity constraint is implemented as an additional reward function:

$$r_{t,2} = \begin{cases} -1, & \text{if } |\boldsymbol{\omega}_B^{BN}| \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

This reward function is used to prevent angular velocities that could be dangerous to the spacecraft and to prevent instability in the numerical simulation. Additionally, it was found that adding this angular velocity limit actually improved the performance of the controller; this could be because the velocity of the reference trajectories generally does not exceed 1 rad/s and, as such, limiting the search space for behavioural policies eases the search for a desirable policy. The total reward provided to the agent is the sum of these two reward functions $r_t = r_{t,1} + r_{t,2}$.

4.4 Implementation Details

4.4.1 Agents

The objective of the tracking SLERP trajectory scenario is to ascertain to what degree LSTM layers can be used to reconstruct temporal information not present in the observation vector. As SLERP trajectories are constant angular velocity, if the observation vector contains the angular velocity $\boldsymbol{\omega}_R^{RN}$ of the reference, it will have Markov property. Four different agent neural network architectures were investigated, as detailed in Table 1. All architectures have as many input neurons as elements in their observation vector, 3 output neurons to provide control torques for each body axis, and use 2 fully-connected dense layers of 64 neurons to parameterise the behavioural policy. Agents 3 and 4 have an additional LSTM layer directly after the input, which provides these agents with the ability to remember some information from previous observation vectors in an internal state. For the SLERP tracking scenario, agents 1, 2, and 3 were trained. For agents 1 and 3, the state vector is modified to remove angular velocities $\boldsymbol{\omega}_R^{RN}$ and $\boldsymbol{\omega}_B^{BR}$. Agent 1

can only observe the instantaneous position of the reference. Agent 3 attempts to reconstruct the missing temporal information by using a layer of 256 LSTM cells.

For SQUAD trajectory tracking, there is no simple way to define a Markov observation vector. The endpoint and support quaternions, as well as the interpolation parameter, could be added to the state vector, but it is assumed in this work that the trajectory to be tracked is not known to the agent. For the SQUAD tracking scenario, agents 1, 2, and 4 were trained. Here agents 1 and 2 use the same architecture as in the SLERP tracking scenario but are retrained in the SQUAD tracking scenario. Agent 4 is an LSTM variant of Agent 2 that attempts to construct temporal information that cannot be observed by Agent 2 due to the non-constant angular velocity of the reference. Agents 1 and 2 were trained using the PPO algorithm, implementation from Stable-Baselines3, and agents 3 and 4 use the recurrent PPO implementation from Stable Baselines3 - Contrib.[13] All agents use the hyperparameters listed in Table 2. In all cases, both the actor and critic networks use the architecture specified in Table 1, except that the critic only has one output neuron corresponding to the estimated value of the current observation. The actor and critic networks are separate and do not share any layers.

Table 1: Agent network architectures

		Agent 1	Agent 2	Agent 3	Agent 4
Layer	Activation	#Neurons	#Neurons	#Neurons	#Neurons
Input	Linear	30	36	30	36
LSTM	Linear	-	-	256	256
Dense	LeakyReLU	64	64	64	64
Dense	LeakyReLU	64	64	64	64
Output	Linear	3	3	3	3

In DRL, the data gathered by the agent while interacting with the environment is dependent on the current behavioural policy employed by the agent. Before training, the policy is initialised randomly. Whether or not the agent is able to learn a successful behavioural policy can be entirely dependent on this initialisation. To ensure that the agent is generally successful, it is trained independently in 5 repeat runs, each making use of a different random seed. The reference trajectories are also varied across the 5 repeat runs.

Table 2: Hyperparameters

# Independent repeat runs	5
# Asynchronous environments	16
Learn steps	5×10^6
Rollout horizon	500
Mini-batch size	125
Epochs	10
Clip range	0.2
Discount factor	0.99
Learning rate	3×10^{-4}
Optimizer	Adam
Evaluation frequency	5000
# Evaluation episodes	20
# Testing episodes	1000

4.4.2 Episode

At the start of every episode, a uniformly-random quaternion[18] is generated for the initial orientation of the spacecraft $\boldsymbol{q}^{BN}$, for the SLERP and SQUAD tracking scenarios the two endpoint quaternions are generated randomly as well as the two support quaternions for the SQUAD. On every control time step, the agent outputs three control torques to be applied. During training, Gaussian white noise is superimposed onto the commanded control torque to ensure sufficient exploration of the state space. During testing and evaluation, the exploration noise is disabled as the agent is no longer learning and does not need to explore. The body torques to be applied are constrained such that each body torque should be less than 0.005 Nm in magnitude. An episode ends when the simulation time reaches the end of the manoeuvre (t=100 s) or if the spacecraft's angular velocity exceeds 1 rad/s. The agents are simultaneously trained in

16 asynchronous environments, which speeds up training and improves stability.[11,16] Every agent is trained for 5×10^6 time steps corresponding to approximately 10000 episodes or 11 days and 14 hours of simulated time. During training, after every 5000 steps, the agents are evaluated for 20 episodes. After training, the best instance (as determined by mean reward during evaluation episodes) of every agent is tested for 1000 episodes to establish the final tracking performance of the agent. The best agent, according to evaluation episode performance, is tested instead of the final agent as training can be destructive, and the final agent (weights), after training, is not necessarily the best-performing instance of that agent.

4.5 Quaternion Rate Feedback Controller

The performance of the DRL attitude controller is compared against that of a Quaternion Rate Feedback (QRF) controller. The QRF controller[14] is implemented for tracking of the reference trajectory:

$$\boldsymbol{L}_B = -K\boldsymbol{\beta}^{BR} - \mathbf{P}\boldsymbol{\omega}_B^{BR} + \boldsymbol{\omega}_B^{BN} \times (\boldsymbol{J}_B\boldsymbol{\omega}_B^{BN}) \tag{20}$$

where K is a scalar gain and $\mathbf{P}$ is a diagonal gain matrix and $\boldsymbol{\beta}^{BR}$ is a three-component vector of the error quaternion $\boldsymbol{q}^{BR}$ between the body and reference orientation. To ensure the controller takes the shortest path towards the target rotation, the sign of $\boldsymbol{\beta}^{BR}$ is switched if the scalar component is less than 0:

$$\boldsymbol{\beta}^{BR} = \begin{cases} (q_1,\, q_2,\, q_3), & \text{if } q_w \geq 0 \\ (-q_1,\, -q_2,\, -q_3), & \text{else} \end{cases} \tag{21}$$

Initial gain values were obtained by linearising the dynamics presented in Eqs. (4) and (10), and then adjusting the gains to be less aggressive to avoid overshoot due to the actuator saturation limit.

$$K = 0.2 \tag{22}$$

$$\mathbf{P} = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.26 \end{bmatrix} \tag{23}$$

5. Results

The average rewards obtained by the agents during training for SLERP trajectory tracking are shown in Fig. 2. Agent 2, which has access to the angular velocity of the reference, learns faster than the other agents and maintains the highest average reward throughout training. Initially, Agent 3 learns slower than the other agents but is then able to achieve significantly higher rewards than Agent 1. The LSTM Layer allowed Agent 3 to infer the angular velocity of the reference trajectory and use this information to obtain better tracking performance than Agent 1. The average rewards obtained during training for SQUAD trajectory tracking are shown in Fig. 3. Here agents 1 and 2 achieve slightly lower average reward than for tracking of SLERP trajectories due to non-constant angular velocity. Agent 4, which is Agent 2 with an LSTM layer, performs worse than Agent 2, showing that, in this case, the LSTM layer did not help the agent infer additional temporal information.

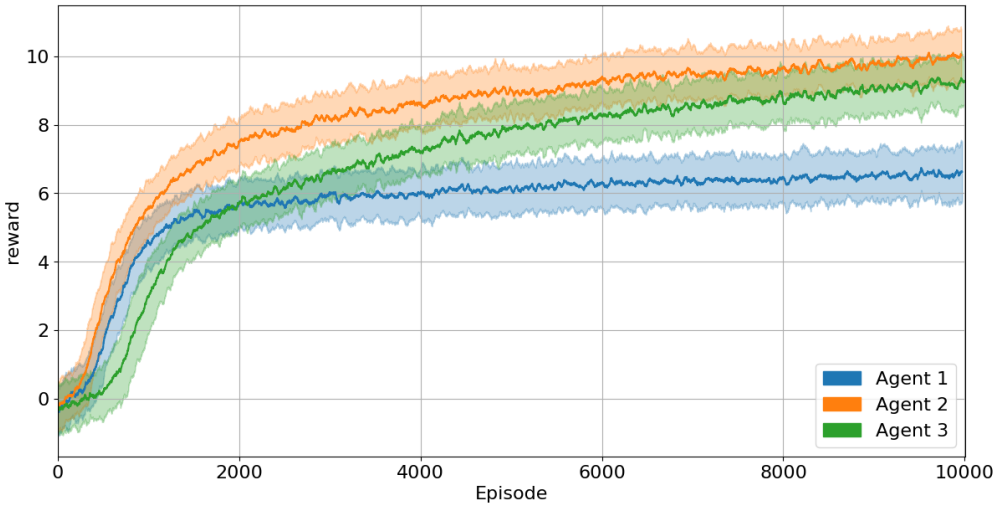

Figure 2: Sliding mean (window size: 40) of average reward (across runs) during training for SLERP trajectory tracking. Shaded area indicates standard deviation.

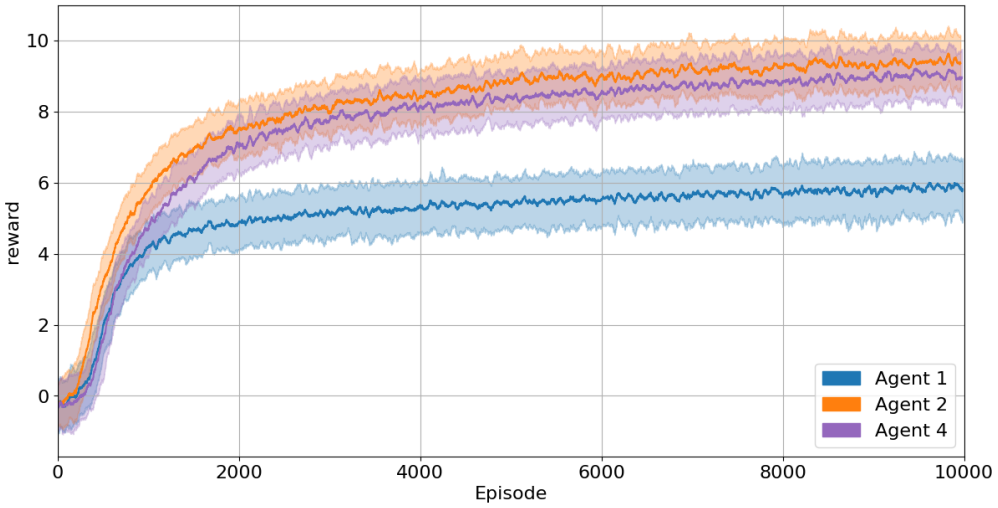

Figure 3: Sliding mean (window size: 40) of reward during training for SQUAD trajectory tracking. Shaded area indicates standard deviation.

As the spacecraft begins an episode in a random orientation, it will generally not start while being aligned with the reference trajectory. The controller must first catch up (slew) with the reference trajectory and then begin tracking it as closely as possible. As a result of this, the tracking error is usually high during the first half of the episode. The performance metric considered is the mean tracking error during the final 50 seconds of the episode. The performance of the DRL attitude controller when tracking SLERP and SQUAD trajectories in 1000 testing episodes is shown in Figs 4 and 5 respectively as violin plots. Violin plots are similar to box plots but instead of showing the interquartile range a rotated kernel density plot is shown on each side. These plots are used to show the distribution of mean pointing errors across episodes. Fig. 4 shows that, across all runs, Agent 3 outperforms Agent 1 but performs worse than Agent 2. This demonstrates that for SLERP tracking, Agent 3 was able to utilise the LSTM layer to infer the angular velocity of the reference trajectory. The tracking performance of Agent 1 is poor, showing the requirement to know the angular velocity of the reference trajectory. Fig. 5 shows that the LSTM layer did not allow Agent 4 to infer additional information about the non-constant angular velocity of SQUAD trajectories. This could be due to the fact that the rate of change of the angular velocity and rotational axis is small compared to the 100 s episode horizon.

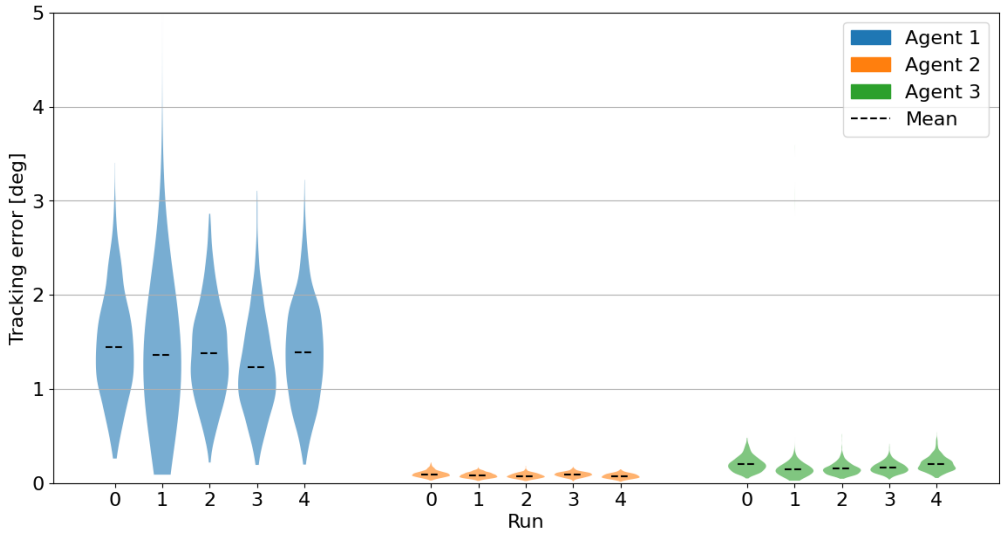

Figure 4: Violin plot of DRL agent mean tracking error during last 50 s of episode in 1000 testing episodes of tracking SLERP trajectories.

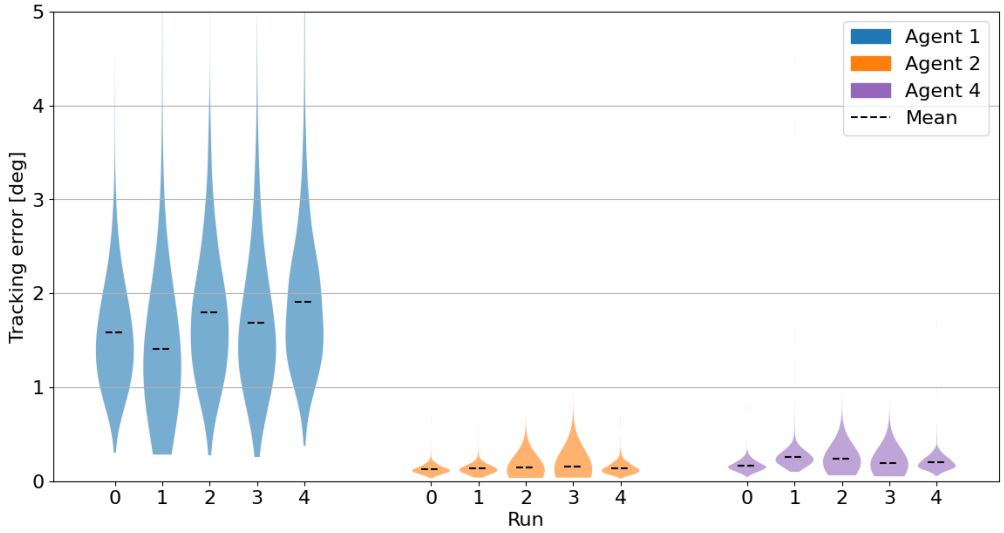

Figure 5: Violin plot of DRL agent mean tracking error during last 50 s of episode in 1000 testing episodes of tracking SQUAD trajectories.

Tables 3 and 4 provide the mean, standard deviation, minimum, and maximum tracking error of the best runs (as defined by mean reward) in the SLERP and SQUAD tracking scenarios, respectively. In both cases, the tracking performance of the QRF controller is also provided. For the SLERP tracking scenario, the QRF controller provides approximately an order of magnitude better tracking performance, with a significantly reduced spread in performance across episodes. Both agents 1 and 3 have a high maximum tracking error; in these particular episodes, the controller was still catching up with the reference at the halfway point in the episode. This could be caused by the fact that the reward function does not explicitly require the controller to catch up to the reference as quickly as possible. The agents are only encouraged to catch up to the reference trajectory in minimum time due to reward discounting. In the SQUAD tracking scenario, the QRF controller outperforms the best DRL agent, but by a lesser margin. The maximum tracking error of Agent 1 is particularly large. It is not clear what exactly caused this; near the end of the episode, the tracking error was reduced to 3.2° with the spacecraft oscillating about the reference in an underdamped fashion. The second-largest tracking error of Agent 1 when tracking SQUAD trajectories was 9.8971°. The QRF controller also has a high maximum tracking error; in this particular episode, the reference trajectory had a rapid angular velocity change near the halfway point, which caused the increased tracking error.

Table 3: Comparison of mean SLERP tracking performance between DRL agent and QRF controller.

Controller	mean($\lvert\phi\rvert$) [deg]	std. dev.($\lvert\phi\rvert$) [deg]	min($\lvert\phi\rvert$) [deg]	max($\lvert\phi\rvert$) [deg]
Agent 1[a]	1.2272	0.4964	0.1918	3.1043[b]
Agent 2[a]	0.0733	0.0250	0.0142	0.1455
Agent 3[a]	0.1480	0.1766	0.0245	3.5897[b]
QRF	0.0087	0.0007	0.0007	0.0107

[a] Best run according to mean reward
[b] Controller had not caught up to reference by $t = 50$ s

Table 4: Comparison of mean SQUAD tracking performance between DRL agent and QRF controller.

Controller	mean($\lvert\phi\rvert$) [deg]	std. dev.($\lvert\phi\rvert$) [deg]	min($\lvert\phi\rvert$) [deg]	max($\lvert\phi\rvert$) [deg]
Agent 1[a]	1.4086	2.4998	0.2826	77.3763[b]
Agent 2[a]	0.1253	0.0563	0.0312	0.7576
Agent 4[a]	0.1636	0.0613	0.0448	0.9613
QRF	0.0881	0.0925	0.0239	2.4770[c]

[a] Best run according to mean reward
[b] Cause unknown
[c] Trajectory had a rapid angular velocity change near $t = 50$ s

Agents 1-3 were retrained but with unit quaternion attitude parameterisation. Specifically, the elements $\mathbf{C}^{BN}$, $\mathbf{C}^{RN}$, and $\mathbf{C}^{BR}$ in Eq. (16) are replaced with $\boldsymbol{q}^{BN}$, $\boldsymbol{q}^{RN}$, and $\boldsymbol{q}^{BR}$. The performance of the quaternion agents is compared against the direction cosine matrix agents in Fig. 6 (note the logarithmic y-axis). For all three agents, the direction matrix attitude parameterisation achieves on average better tracking error than the quaternion agents. Agent 1 is the least impacted by this change, but the performance of Agents 2 and 3 is severely impacted. The reason that the quaternion agents underperform is due to the two-to-one mapping of quaternions to rotations. Specifically, both quaternions $\boldsymbol{q}^{BR}$ and $-\boldsymbol{q}^{BR}$ represent the same physical orientation, and as such, the reinforcement learning agent needs to learn when to target which of the two quaternions as moving to the other one will require a rotation by 180°. The sign of the error quaternion $\boldsymbol{q}^{BR}$ will switch if either $\boldsymbol{q}^{BN}$ or $\boldsymbol{q}^{RN}$ switch sign.

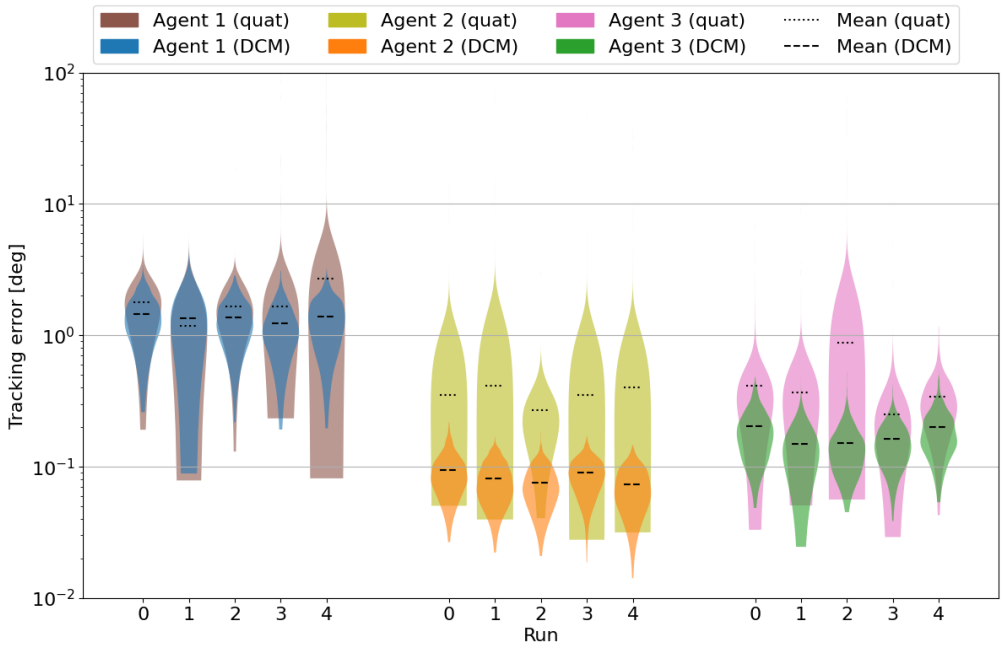

Figure 6: Violin plot of SLERP tracking performance in 1000 testing episodes of agents 1-3 when trained with a quaternion attitude parameterisation.

6. Conclusion

This paper investigated Deep Reinforcement Learning (DRL) for attitude tracking of SLERP and SQUAD trajectories. For SLERP trajectories, the DRL controller that had access to the observation vector containing the angular velocity significantly outperformed the agent that only had access to the position of the reference trajectory. The DRL attitude controller was able to utilise recurrent Long Short-Term Memory (LSTM) layers to significantly improve tracking performance when the observation vector contained only the orientation of the reference trajectory and not the associated angular velocity. For SQUAD trajectory tracking, an additional LSTM layer and angular velocity measurements provided slightly worse tracking performance than the agent that did not have an LSTM layer. The latter may be due to the fact that the angular velocity during the SQUAD changes slowly and any additional information that the agent could infer about how the angular velocity will change only provides a small benefit, which might have been outweighed by the increased difficulty of learning a recurrent policy. For both SLERP and SQUAD trajectories, the DRL-based attitude controller is outperformed by a traditional Quaternion-Rate Feedback (QRF) controller. It was also found that DRL agents that utilise a quaternion-based attitude parameterisation in the observation vector perform significantly worse than DRL agents that use flattened direction cosine matrices to parameterise orientation.

Replacing the traditional attitude control pipeline with a DRL agent provides poorer performance and results in the loss of stability guarantees. In future research, the combination of state estimators to infer temporal information, QRF controller for high-frequency control, and DRL for gain scheduling of the QRF controller will be investigated. This hierarchical control structure will be investigated for more complex problems, including full attitude and orbit control, highly-variable inertia spacecraft attitude control, and multi-spacecraft attitude and orbit control.

7. Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council [grant number 2605755].

References

[1] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[2] Thomas A. W. Dwyer and Hebertt Sira-Ramirez. Variable-structure control of spacecraft attitude maneuvers. *Journal of Guidance, Control, and Dynamics*, 11(3):262–270, 1988.

[3] Jacob Elkins, Rohan Sood, and Clemens Rumpf. Adaptive continuous control of spacecraft attitude using deep reinforcement learning. In *Proceedings of 2020 AAS/AIAA Astrodynamics Specialist Conference*, 08 2020.

[4] Jacob Elkins, Rohan Sood, and Clemens Rumpf. Autonomous spacecraft attitude control using deep reinforcement learning. In *71st International Astronautical Congress (IAC) 2020*, CyberSpace Edition, 10 2020.

[5] Brian Gaudet, Roberto Furfaro, Richard Linares, and Andrea Scorsoglio. Reinforcement metalearning for interception of maneuvering exoatmospheric targets with parasitic attitude loop. *Journal of Spacecraft and Rockets*, 58(2):386–399, 2021.

[6] Brian Gaudet, Richard Linares, and Roberto Furfaro. Deep reinforcement learning for six degree-of-freedom planetary landing. *Advances in Space Research*, 65(7):1723–1741, 2020.

[7] Brian Gaudet, Richard Linares, and Roberto Furfaro. Six degree-of-freedom hovering over an asteroid with unknown environmental dynamics via reinforcement learning. In *AIAA Scitech 2020 Forum*, page 0953, 2020.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[13] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[14] Hanspeter Schaub and John L Junkins. *Analytical mechanics of space systems Fourth Edition*. AIAA, 2018.

[15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[17] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[18] Ken Shoemake. Uniform random rotations. In *Graphics Gems III (IBM Version)*, pages 124–132. Elsevier, 1992.

[19] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[20] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[21] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[22] S. R. Vadali. Variable-structure control of spacecraft large-angle maneuvers. *Journal of Guidance, Control, and Dynamics*, 9(2):235–239, 1986.

[23] Fnu Vedant, James Allison, Matthew West, and Alexander Ghosh. Reinforcement learning for spacecraft attitude control. In *70th International Astronautical Congress (IAC) 2019*, Washington DC, United States of America", 10 2019.

[24] Yuejiao Wang, Zhong Ma, Yidai Yang, Zhuping Wang, and Lei Tang. A new spacecraft attitude stabilization mechanism using deep reinforcement learning method. 2017.

[25] B. Wie, H. Weiss, and A. Arapostathis. Quarternion feedback regulator for spacecraft eigenaxis rotations. *Journal of Guidance, Control, and Dynamics*, 12(3):375–380, 1989.

[26] Bong Wie and Peter M. Barba. Quaternion feedback for spacecraft large angle maneuvers. *Journal of Guidance, Control, and Dynamics*, 8(3):360–365, 1985.