3-2021

Deep Reinforcement Learning Applied to Spacecraft Attitude Control and Moment of Inertia Estimation via Recurrent Neural Networks

Nathaniel A. Enders

DEEP REINFORCEMENT LEARNING APPLIED TO SPACECRAFT ATTITUDE CONTROL AND MOMENT OF INERTIA ESTIMATION VIA RECURRENT NEURAL NETWORKS

THESIS

Nathaniel Enders, Second Lieutenant, USSF

AFIT-ENY-MS-21-M-298

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This is an academic work and should not be used to imply or infer actual mission capability or limitations.

AFIT-ENY-MS-21-M-298

DEEP REINFORCEMENT LEARNING APPLIED TO SPACECRAFT ATTITUDE CONTROL AND MOMENT OF INERTIA ESTIMATION VIA RECURRENT NEURAL NETWORKS

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Astronautical Engineering

Nathaniel Enders, BS

Second Lieutenant, USSF

March 25, 2021

AFIT-ENY-MS-21-M-298

DEEP REINFORCEMENT LEARNING APPLIED TO SPACECRAFT ATTITUDE CONTROL AND MOMENT OF INERTIA ESTIMATION VIA RECURRENT NEURAL NETWORKS

THESIS

Nathaniel Enders, BS
Second Lieutenant, USSF

Committee Membership:

Maj Joshuah A. Hess, PhD
Chair

Maj Costantinos Zagaris, PhD
Member

Richard G. Cobb, PhD
Member

Maj Joseph A. Curro, PhD
Member

Abstract

On orbit, there are situations where the inertia of a spacecraft may be unknown. For example, if a spacecraft were to dock with an object of unknown mass, the combined inertia could be unknown. Because typical methods of spacecraft attitude control require accurate models of the spacecraft, having unknown inertia can make attitude control difficult. This study investigated two distinct problems related to unknown spacecraft inertia where the proposed solution to both problems is machine learning. The first problem explored the use of a recurrent neural network to estimate spacecraft moments of inertia using angular velocity measurements. Initial results showed that, for the configuration examined, the neural network can estimate the moments of inertia when there is a known external torque. The second problem trained a reinforcement learning agent, via proximal policy optimization, to control the attitude of a spacecraft, where the agent had no knowledge of the underlying dynamics or spacecraft model. The results demonstrated that reinforcement learning may be a viable option for guidance and control solutions where the spacecraft model may be unknown. The trained agents displayed a degree of autonomy with their ability to recover from events never experienced in training. These proof-of-concept machine-learning solutions provide a potential path forward for creating autonomous space systems.

Acknowledgements

I would first like to offer a huge thanks to my advisor Maj Hess for guiding me through the most challenging 18 months of my academic career and for being a role model and inspiration to me. It is no understatement to say that without the feedback, critiques, instruction, and advice from Maj Hess, this research would not have been completed. I also offer much thanks to Maj Zagaris, Dr. Cobb, and Maj Curro, who in instructing me in the ways of aerospace and computer science, have inspired a love of learning and an appreciation for the depth of their fields. Thank you to my committee in general for your hard work, your feedback, and your support.

To all of the teachers and instructors who have taught me over the years, thank you for encouraging me to not only learn, but to enjoy learning. To all of my friends and family, thank you for your unwavering love and support. A special thanks to my parents for always being there for me and for instilling the work ethic and character that brought me to where I am today.

Finally, I would like to thank my wife and best friend. Your support and encouragement has meant the world to me. Thank you for listening to me rant, for being a sounding board for ideas, and for helping me find a balance between work and life.

Nathaniel Enders

Table of Contents

Page

Page

List of Figures

Figure Page

List of Tables

List of Acronyms

AI artificial intelligence.

CCS Cartesian coordinate system.

CNN convolutional neural network.

D4PG distributed distributional deep deterministic policy gradient.

DCM direction cosine matrix.

DDPG deep deterministic policy gradient.

DOF degree-of-freedom.

LQR linear quadratic regulator.

LSTM long short-term memory.

MAE mean absolute error.

MAPE mean absolute percentage error.

MDP Markov decision process.

ML machine learning.

MOI moment of inertia.

MPC model predictive control.

MSE mean squared error.

PD proportional derivative.

PI proportional integral.

PID proportional integral derivative.

PPO proximal policy optimization.

ReLU rectified linear unit.

RL reinforcement learning.

RNN recurrent neural network.

SSA space situational awareness.

TD temporal difference.

TRPO trust region policy optimization.

UAV unmanned aerial vehicle.

List of Symbols

a action.

A arbitrary rotation matrix.

$\hat{A}$ advantage estimate.

A_t the action at the current time step.

$\vec{a}$ action vector (normalized).

$\vec{B}$ magnetic flux density.

$\dot{\vec{B}}$ time rate of change of magnetic flux density.

$\vec{b}$ bias vector for neural network.

C_{BA} direction cosine matrix, rotate from frame B to frame A.

$\hat{e}$ Euler axis.

$e(t)$ tracking error.

$\dot{e}(t)$ time derivative of the tracking error.

g arbitrary activation function for neural network.

$h_i^{(j)}$ hidden layer unit, where i is the unit, and j is the layer.

$\vec{h}$ hidden layer vector.

I_3 a 3×3 identity matrix.

J inertia matrix.

$J(\pi_\theta)$ reinforcement learning performance objective.

J_{avg} average moment of inertia $((J_x + J_y + J_z)/3$.

J_x principal moment of inertia about the x axis.

J_y principal moment of inertia about the y axis.

J_z principal moment of inertia about the z axis.

k gain for "B-dot" steering law.

K number of epochs.

k_1 first Smelt parameter.

k_2 second Smelt parameter.

k_3 third Smelt parameter.

K_d derivative gain.

K_i integral gain.

K_i random sample drawn from normal distribution.

K_p proportional gain.

L surrogate loss function.

M minibatch size.

$\vec{m}_m$ magnetic moment of a spacecraft.

$m(t)$ measurements from the plant.

$\vec{m}$ magnetic moment produced by magnetic torquers.

N number of actors.

$\mathcal{N}$ normal distribution.

p_c penalty on the sum of the control.

q quaternion, $q = [q_1,\ q_2,\ q_3,\ q_4]^T$.

Q action-value function.

q_d desired (or commanded) quaternion.

q_e error quaternion.

$Q_\theta(a,\ s)$ Q-function parameterized by θ with state and action inputs.

$[\bar{q}^\times]$ skew-symmetric matrix from quaternion vector.

r reward.

R_c distance from center of Earth to spacecraft center of mass.

$\vec{R}_c$ vector originating at center of Earth, pointing to spacecraft center of mass.

r_f^- value of reward field before taking step.

r_f^+ value of reward field after taking step.

$r(t)$ reference signal.

R_t the reward at the current time step.

s state.

S_t the state at the current time step.

T number of timesteps.

u sum of the absolute values of the control vector.

U uniform distribution.

$u(t)$ control effort.

$\vec{u}$ control vector.

W weighting matrix for neural network.

$w(t)$ external disturbances.

$\vec{w}_{max}$ maximum allowable angular velocity.

X inputs to a function or neural network.

x_i measurement or input to neural network.

x_i random samples drawn from uniform distribution.

$\vec{x}$ vector input to neural network.

y scalar output of neural network.

Y outputs of a function or neural network.

α Euler angle.

β_0 intercept of a line.

β_1 slope of a line.

θ policy or Q-function parameters.

θ_i uniformly random angle.

μ mean.

$\mu_\oplus$ gravitational parameter of the Earth.

π the policy, rules for taking action.

π_θ policy, π, parameterized by θ.

$\pi_\theta(a|s)$ policy, π, parameterized by θ and returns an action, a, given the state, s.

σ standard deviation.

σ^2 variance.

$\vec{\tau}$ torque vector.

$\vec{\tau}_c$ control torque.

$\vec{\tau}_d$ disturbance torque.

$\vec{\tau}_{gg}$ gravity gradient torque.

$\vec{\tau}_m$ magnetic torque.

$\vec{\omega}$ angular velocity vector.

$\vec{\omega}_d$ desired angular velocity.

$\dot{\vec{\omega}}$ angular acceleration vector.

$[\vec{\omega}^\times]$ skew-symmetric matrix made from angular velocity.

DEEP REINFORCEMENT LEARNING APPLIED TO SPACECRAFT ATTITUDE CONTROL AND MOMENT OF INERTIA ESTIMATION VIA RECURRENT NEURAL NETWORKS

I. Introduction

1.1 Motivation

As the United States Space Force establishes itself as the sixth service branch, it will continue to pursue the advancement of space technologies as laid out in the Space Force doctrine [5]. These advancements span a wide range of diverse fields, one of which is autonomy. In accordance with both the National Security Strategy (NSS) [6] and the National Defense Strategy (NDS) [7], the United States will "invest broadly in ... autonomy, artificial intelligence, and machine learning" [7]. According to the U.S. Air Force 2030 Science and Technology Strategy, "to realize the potential of artificial intelligence, the Air Force scientific and technical enterprise must push well beyond developed commercial applications" [8].

Russell and Norvig describe autonomy as the ability of an agent to learn what it can to compensate for partial or incorrect prior knowledge [9]. Alternatively, an agent that *lacks* autonomy would rely on the prior knowledge of its designer rather than on its own perceptions [9]. The key concepts at play are *learning*, and *adaptation*. An autonomous agent can *learn* from its current perceptions and *adapt* to environments it has never seen before without ever being explicitly instructed how to behave. The difference between the often conflated concepts of autonomy and automation is that

automation does not require learning and adaptation, although an automated agent may still be able to operate without human input (see [10] for a thorough distinction).

Because autonomy deals with learning, it is no surprise that the field of machine learning is deeply intertwined with the field of autonomy. There are many examples of autonomous systems being developed from self-driving cars, to autonomous pilots. In August of 2020, an artificial intelligence (AI) algorithm controlling an F-16 defeated a human US Air Force pilot 5-0 in a simulated dogfight for DARPA's Alpha Dogfight Trials [11]. Another DARPA project called Physics of Artificial Intelligence (PAI) aims to incorporate physics, mathematics, and prior knowledge relevent to DoD application domains with sparse data into artificial intelligence systems [12]. A third DARPA project called Techniques for Machine Vision Disruption (TMVD) is investigating the disruption of machine vision systems without detailed knowledge of their internal architecture or how they were trained [13]. These DARPA projects highlight the progress made in autonomy thus far and the motivation behind why autonomy is desirable in the first place: to intelligently perform complex missions in challenging environments with greatly reduced need for human intervention [14].

Another domain where autonomy can be applied is in spacecraft attitude control, which deals with controlling the orientation of a spacecraft. Most of spacecraft attitude control is already automated in the sense that there is an algorithm telling the actuators how much torque to generate to achieve a specific orientation with no human input other than perhaps the desired orientation. These automated algorithms are often not autonomous because they rely on pre-determined models of the spacecraft. If there are large enough deviations from the spacecraft model that were unaccounted for, then the spacecraft may no longer be controllable. However, a greater degree of autonomy could be more robust to changes because the spacecraft could adapt to changes in the model or dynamics and still function.

1.2 Research Overview

In line with the US Air Force's 2030 Science and Technology Strategy, the NSS, the NDS, and national calls for research into autonomy, this thesis aims to investigate the use of machine learning to allow for a greater degree of autonomy in spacecraft attitude control. When controlling a spacecraft's orientation, or attitude, most control techniques require an accurate model of the spacecraft being controlled, with a few techniques allowing for some uncertainty in model parameters. One set of parameters that must be modeled is the inertia matrix of the spacecraft, which is a measure of how the mass of a spacecraft is distributed. On orbit, there are situations where the inertia of a spacecraft may be unknown. For example, if a spacecraft were to dock with an object of unknown mass, the combined inertia may be unknown. Because most methods of spacecraft attitude control require accurate models of the spacecraft, having unknown inertia can make attitude control difficult. This study investigated two distinct problems related to unknown spacecraft inertia where the proposed solution to both problems is machine learning. These two problems will be referred to as Problem A and Problem B.

1.2.1 Problem A: Inertia Estimation via Neural Networks.

One possible solution to controlling a spacecraft with an unknown inertia matrix is to estimate the inertia and then use typical control solutions such as a linear quadratic regulator (LQR), proportional integral derivative (PID) control, or model predictive control (MPC). Depending on the scenario, it may be possible to use an adaptive control technique, or an estimation technique like a Kalman filter. However, the solution proposed in this thesis is to use machine learning to estimate the inertia matrix of a spacecraft. Using recurrent neural networks may be a viable inertia estimation technique for some scenarios.

1.2.2 Problem B: Attitude Control via Reinforcement Learning.

Besides estimating the inertia values, another possible solution to control a spacecraft with unknown inertia is to use machine learning to control the spacecraft with no knowledge of the spacecraft model or dynamics. The goal of Problem B is to use deep reinforcement learning, a type of machine learning, to control the attitude of a spacecraft. One possible benefit of using reinforcement learning is the potential of achieving a generalized agent that can control a wide range of spacecraft with different values for inertia and maximum torque outputs. The implication is that the same algorithm could potentially control a spacecraft both before and after a collision event where the spacecraft either gained or lost an unknown amount of mass. Because the algorithm does not require knowledge of the underlying model of the spacecraft, the reinforcement learning agent could potentially control the spacecraft regardless of what the model is and whether or not the model is changing. In addition to a collision event, other scenarios may also benefit from having a reinforcement learning agent as a controller. For example, spacecraft with articulated appendages or robotic arms could have varying inertia, and the change in inertia from fuel use could also warrant a reinforcement learning controller.

1.3 Assumptions and Limitations

For Problem A, the inertia matrix being estimated and governing the dynamics is always expressed in the principal frame, meaning there are three nonzero moments of inertia, and three products of inertia equal to zero. The first part of Problem A uses data generated with no external torques. The second and third parts of Problem A use data which only considers a gravity gradient torque and a magnetic torque. There is no consideration of aerodynamic drag, solar radiation pressure, third-body

effects, etc. Problem A also assumes perfect sensors, resulting in perfect knowledge of the angular velocity and torques (when present).

Problem B uses randomly generated inertia matrices which are also always expressed in the principal frame instead of in a general body frame. There are no external disturbance torques considered in any part of Problem B. It also assumed the maximum allowable angular velocity requested by the algorithm is a function of the average of the moments of inertia, as explained in Section 3.5.1.2. Problem B also assumes perfect sensors, meaning there is no noise present in the state information.

1.4 Summary of Results

The results of Problem A show that a recurrent neural network can be trained to estimate the moments of inertia of a spacecraft when there is an external torque acting on the spacecraft. Additionally, the neural networks were able to estimate the relative moment of inertia ratios with high accuracy regardless of external torque or knowledge thereof.

Problem B investigated the use of a reinforcement learning algorithm to control the attitude of a spacecraft. The resulting agents were able to successfully control the spacecraft to the desired orientations. The agents were successful despite having no knowledge of the spacecraft inertia or attitude dynamics. One additional finding was the agents were able to control the spacecraft when subject to a random change in inertia and angular velocity halfway through the simulation. This finding is very promising because the agents were never trained to handle abrupt changes to inertia and angular velocity. The ability of the agents to handle unseen scenarios suggests that the agents have achieved some degree of autonomy.

1.5 Thesis Overview

The remainder of this thesis consists of four chapters. Chapter II explains much of the information required to understand the solutions to Problems A and B. Chapter II also covers previous work related to inertia estimation and to reinforcement learning applied to aerospace. Chapter III covers the problems in more detail along with the hardware and software used to solve the problems. Chapter III also details the methodology used to solve Problems A and B. Chapter IV presents and discusses the results for Problems A and B, and Chapter V draws some conclusions based on the results.

II. Background

2.1 Overview

In order to answer the questions presented in the Introduction, some background is required in a few different subjects. The first is attitude kinematics, dynamics, and control, which describes the orientation of a body, how the orientation changes over time, and how to control the orientation. The next subject is machine learning and neural networks, which are techniques for making a program able to learn from data. Building on top of machine learning is reinforcement learning, which is a process for a program to learn how to interact with an environment. Next, a literature review is presented, showing related research in estimating spacecraft moments of inertia (MOI) and in the use of reinforcement learning for attitude control problems.

2.2 Attitude Kinematics, Dynamics, and Control

The attitude of a body describes the orientation of that body in three-dimensional space [15]. When dealing with spacecraft, there must be a way to both describe and control where the spacecraft is pointing. There are several different parameterizations for a spacecraft's attitude, and there are also many different ways of controlling a spacecraft's attitude.

2.2.1 Attitude Kinematics.

Kinematics is the study of motion, and attitude kinematics provides a method for describing the rotational motion of a rigid body [16]. There are several different ways to parameterize attitude such as a direction cosine matrix (DCM), Euler axis/angle, Euler angles, quaternions (or Euler parameters), Rodrigues parameters, and modified Rodrigues parameters. The DCM, Euler's eigenaxis rotation theorem, and the

quaternion will be discussed in more detail. The SO(3) configuration space is the set of all orthogonal 3×3 matrices whose determinant is $+1$ [17]. All attitudes are contained within the SO(3) configuration space and have three degrees of freedom. A DCM can be defined as a 3×3 orthonormal matrix whose inverse is its transpose [18]. The notation for a DCM is shown in Equation (2.1),

$$C_{BA} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \tag{2.1}$$

which describes the orientation of reference frame B relative to reference frame A [19]. A DCM is the most fundamental way of describing an orientation, but six of the nine parameters are redundant because only three parameters are required to describe a reference frame orientation [20].

Another attitude parameterization comes from Euler's eigenaxis (or principal) rotation theorem. In essence, this theorem states that the transformation from any initial orientation to any final orientation can be described as a single rotation of a principal angle, α, about the principal axis, $\hat{e}$ [20]. While Euler's eigenaxis attitude parameterization has fewer parameters than a DCM, it comes at the cost of non-uniqueness because the sets $(\hat{e},\ \alpha)$ and $(-\hat{e},\ -\alpha)$ describe the same orientation [20]. Euler's eigenaxis attitude parameterization also contains singularities where the angle, α, is equal to $\alpha = 0,\ \pm 180^o,\ \pm 360^o,\ \ldots$ [19]. With four parameters describing the attitude, the one constraint is that the axis must be a unit vector [19].

A third attitude parameterization consists of the four Euler parameters or unit quaternion. Equation (2.2) shows the notation for the unit quaternion, where $\bar{q}$ represents the vector component of the quaternion and consists of the first three

components, and q_4 represents the scalar part of the quaternion.

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \bar{q} \\ q_4 \end{bmatrix} \tag{2.2}$$

This parameterization is nonsingular but also not unique because the attitude described by the quaternion, q, is equivalent to the attitude described by the quaternion, $-q$ [20]. It is also important to note that the unit quaternion is constrained by the following relationship:

$$||q|| = q_1^2 + q_2^2 + q_3^2 + q_4^2 = \bar{q}^T \bar{q} + q_4^2 = 1 \tag{2.3}$$

The quaternion described by a positive q_4 represents the orientation through the shortest single axis rotation, while the quaternion with a negative q_4 represents the orientation through the longest single axis rotation [20].

Some algorithms make use of an error quaternion to represent the difference between one orientation and a desired orientation. The error quaternion between two unit quaternions must also maintain a unit norm constraint, so the two quaternions cannot simply be subtracted. Instead, the error quaternion, q_e, between some quaternion, q, and the desired quaternion, q_d, is given by Equation (2.4),

$$q_e = q \otimes q_d^* \tag{2.4}$$

where the operator $\otimes$ denotes quaternion multiplication [17]. The conjugate of the desired quaternion, q_d^*, can be expressed as in Equation (2.5) [17].

$$q_d^* = \begin{bmatrix} -\bar{q}_d \\ q_{d4} \end{bmatrix} \tag{2.5}$$

The quaternion multiplication can be performed using matrix multiplication as shown in Equation (2.6)

$$q_e = \begin{bmatrix} q_4 I_3 - [\bar{q}^\times] & \bar{q} \\ -\bar{q}^T & q_4 \end{bmatrix} \begin{bmatrix} -\bar{q}_d \\ q_{d4} \end{bmatrix} \tag{2.6}$$

where $[\bar{q}^\times]$ is the skew-symmetric matrix made up of the vector part of the quaternion, and I_3 is a 3×3 identity matrix [17].

$$[\bar{q}^\times] = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{2.7}$$

2.2.2 Attitude Dynamics.

Whereas attitude kinematics is concerned with describing a body's orientation, attitude dynamics describes how a body's orientation changes over time [19]. Equation (2.8) shows the general attitude dynamics equation in the body-fixed Cartesian coordinate system (CCS),

$$J\dot{\vec{\omega}} + [\vec{\omega}^\times] J\vec{\omega} = \vec{\tau} \tag{2.8}$$

where J is the inertia matrix, $\vec{\omega}$ is the angular velocity, $\dot{\vec{\omega}}$ is the angular acceleration, and $\vec{\tau}$ is the torque [20]. The matrix, $[\vec{\omega}^\times]$ is a skew-symmetric matrix composed of

the angular velocity components:

$$[\vec{\omega}^{\times}] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{2.9}$$

If the body-fixed reference frame is aligned with the spacecraft's principal-axis frame, then Euler's rotational equations of motion become

$$J_x\dot{\omega}_x - (J_y - J_z)\omega_y\omega_z = \tau_x \tag{2.10a}$$

$$J_y\dot{\omega}_y - (J_z - J_x)\omega_z\omega_x = \tau_y \tag{2.10b}$$

$$J_z\dot{\omega}_z - (J_x - J_y)\omega_x\omega_y = \tau_z \tag{2.10c}$$

where J_x, J_y, J_z are the principal moments of inertia, and τ_x, τ_y, and τ_z are the torques about each of the principal axes [19].

There are several environmental sources of torque that can affect a spacecraft's attitude, including gravitational, aerodynamic, radiation, and magnetic torques [18]. One example of an external disturbance torque is the gravity gradient torque, which results from a nonuniform gravitational field causing a torque about a spacecraft's center of mass [18]. Equation (2.11) shows the expression for gravity gradient torque, $\vec{\tau}_{gg}$,

$$\vec{\tau}_{gg} = \frac{3\mu_{\oplus}}{{R_c}^5}\vec{R}_c \times J\vec{R}_c \tag{2.11}$$

where $\mu_{\oplus}$ is the gravitational parameter of the Earth, $\vec{R}_c$ is the vector originating at the center of the Earth and pointing to the spacecraft center of mass, and J is the spacecraft inertia matrix [18].

Another environmental disturbance torque comes from the interaction of Earth's magnetic field with the spacecraft's magnetic moment [18]. Equation (2.12) shows

the expression for magnetic torque, $\vec{\tau}_m$,

$$\vec{\tau}_m = \vec{m}_m \times \vec{B} \tag{2.12}$$

where $\vec{m}_m$ is the net magnetic moment of the spacecraft, and $\vec{B}$ is the external magnetic flux density [18].

2.2.3 Attitude Control.

Once the orientation of a spacecraft and how the orientation changes can be described, the next logical step would be to describe how to control the orientation of the spacecraft. In essence, the general torque term found in Euler's rotational equations of motion changes from the general torque, $\vec{\tau}$, to the sum of control torques, $\vec{\tau}_c$, and disturbance torques, $\vec{\tau}_d$. The general rotational equation of motion then becomes:

$$J\dot{\vec{\omega}} + [\vec{\omega}^{\times}]J\vec{\omega} = \vec{\tau}_c + \vec{\tau}_d \tag{2.13}$$

The control torque, $\vec{\tau}_c$, comes from on-board actuators, such as reaction wheels or control moment gyroscopes, that exchange momentum with the spacecraft to change or maintain the orientation of the spacecraft. The challenge is in determining how much torque is required about each axis at any given time, and this is solved by a control algorithm, such as LQR, MPC, PID, etc.

Most control algorithms are based on the idea of feedback [21]. Figure 2.1 shows a generic control system without feedback [21].

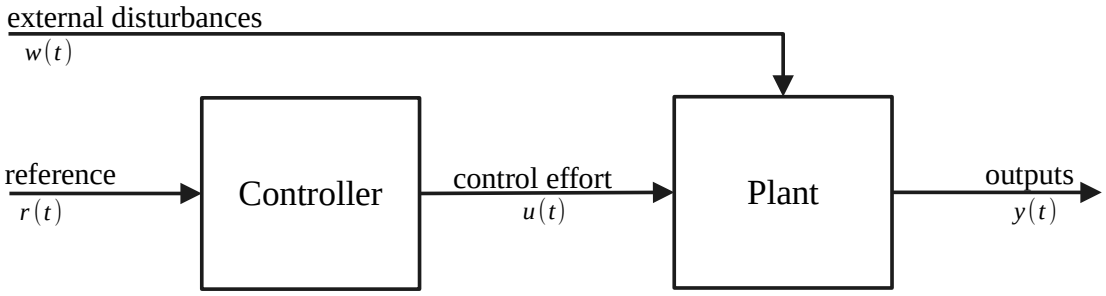

Figure 2.1. Generic open loop block diagram

The controller receives a reference signal (or desired state), $r(t)$, and determines the appropriate control effort, $u(t)$. The plant, or system, then takes the control effort and external disturbances, $w(t)$, and produces an output. Because there is no feedback, the system has no way of correcting its behavior if the output does not match the reference. By taking the output of the plant and using it as an input to the controller, the system can better track the reference. Figure 2.2 shows the same system as Figure 2.1, but with feedback [1].

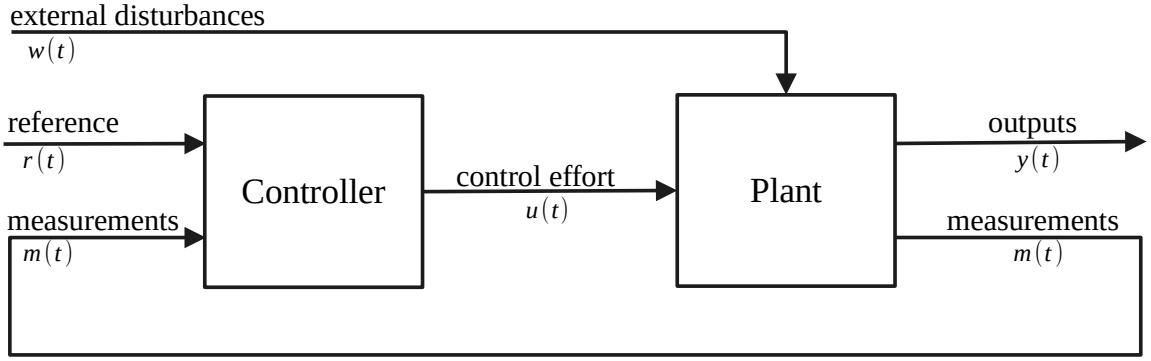

Figure 2.2. Generic closed loop block diagram [1]

In some cases instead of controlling the state to match the reference directly, the control algorithm will instead regulate the error between the reference and the measurements to zero. In other words, the error, $e(t) = r(t) - m(t)$, will be driven to zero instead of driving the measurements to the reference. The effect is often the same, but it can be easier to interpret.

Whichever control algorithm is used will need to calculate the appropriate control effort, $u(t)$, and to do this, a control law is needed. One example is the PID controller, which is made up of a combination of a proportional, integral, and derivative control components [22]. The PID controller can be expressed in the time domain as a function of the error, $e(t)$, as shown in Equation (2.14).

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t_f} e(t)\ dt + K_d \dot{e}(t) \tag{2.14}$$

where K_p is the proportional gain, K_i is the integral gain, and K_d is the derivative gain [22]. The amount of control calculated in Equation (2.14) is proportional to the error, $e(t)$, proportional to the total error over time, $\int_{t_0}^{t_f} e(t)\ dt$, and proportional to the change in error, $\dot{e}(t)$. When the model of the system being controlled is known, the values of the gains can often be calculated to meet the desired specifications, but when the system model is unknown, the gains often require manual tuning [22]. It is also possible to use a subset of the gains to create either a proportional integral (PI), proportional derivative (PD), or just a proportional controller [22].

In some cases, instead of tracking a particular trajectory, a control algorithm may be used to reduce a spacecraft's absolute angular velocity, also known as detumbling. One method for detumbling a spacecraft is to use magnetic torquers along with the "B-dot" steering law. Equation (2.15) shows this steering law,

$$\vec{m} = -\frac{k}{||\vec{B}||}\dot{\vec{B}} \tag{2.15}$$

where $\vec{m}$ is the magnetic dipole, k is some positive gain that can be tuned, and $\dot{\vec{B}}$ is the time rate of change of the magnetic flux density [17]. To calculate the resulting torque produced using this steering law, Equation (2.12) can be used.

2.3 Machine Learning and Neural Networks

Artificial intelligence (AI) is a broad field with a diverse set of subfields such as search and knowledge representation as well as perception and machine learning [9]. At the heart of an intelligent system is an agent. Russell and Norvig define an intelligent agent as one that takes the best possible action in a given situation, where the definition of "best" is dependent on the situation [9]. This definition shares similarities with optimal control theory, where the goal is often to find a solution which both satisfies constraints and minimizes some cost [23]. Although optimal control and AI share similar goals, AI is a much broader field, applying to problems which are more difficult to formulate mathematically such as speech or vision [9].

2.3.1 Machine Learning Overview.

Figure 2.3 shows the basic relationship among artificial intelligence, machine learning, and deep learning. In general, machine learning requires input data, expected outputs, and a measurement of the performance of the algorithm [2]. With enough input data, the model will learn a way to map the input to the expected output while minimizing or maximizing the performance measure. Linear regression is one of the simplest machine learning algorithms because the only parameters that must be learned are the slope, β_1, and the intercept, β_0, of the equation $Y \approx \beta_0 + \beta_1 X$. With a large enough dataset, better values for β_0 and β_1 can be determined, which yield more accurate predictions for the output, Y, within the range of the training set [24].

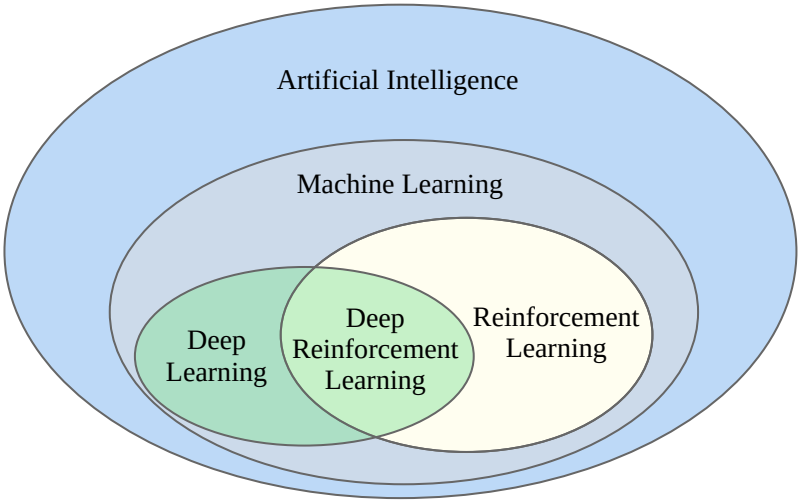

Figure 2.3. Deep learning is a subset of machine learning, which itself is a subset of artificial intelligence [2]

2.3.2 Neural Network Architecture.

Practitioners in the field of machine learning (ML) employ a variety of techniques ranging from linear regression to neural networks. Although many ML tasks can be solved with simpler methods like linear regression, the complexity of some problems require more powerful techniques, such as neural networks. The purpose of neural networks is often described as being a nonlinear function approximator [25]. In fact, a neural network with a single sufficiently large hidden layer can approximate any continuous function [26]. However, in practice, the universal approximation property of neural networks is achieved much more easily by composing many smaller layers rather than a single large layer [3]. Figure 2.4 shows a simplified graph layout of a fully-connected feedforward neural network.

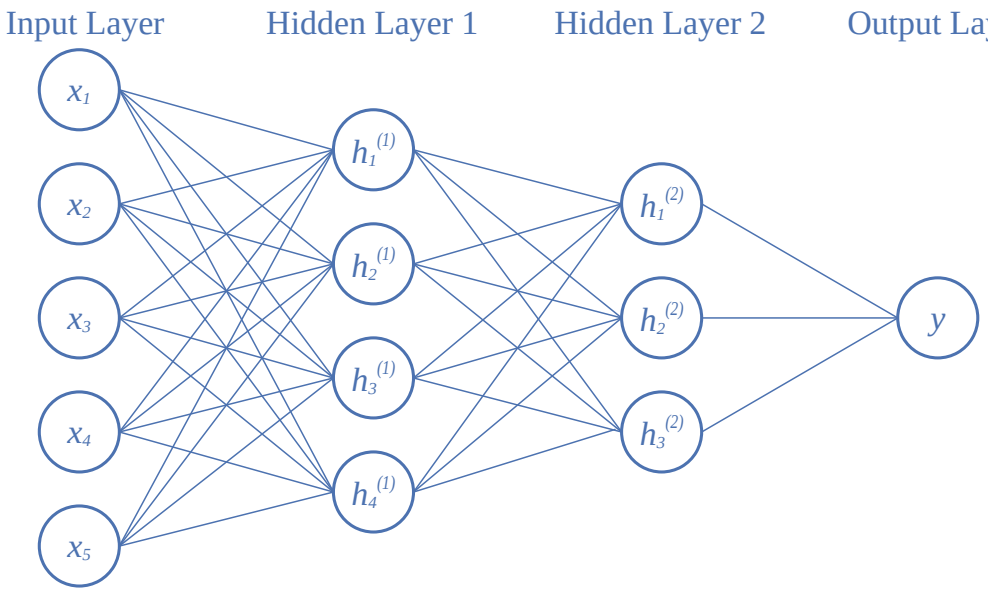

Figure 2.4. A generic example of a feedforward neural network [3]

This example consists of four layers: an input layer, two hidden layers, and an output layer. Each layer contains a set of units or nodes, and for the input layer, each node represents a different input. This example has only a single output node, which means that for a set of five measurements, x_i, the neural network is predicting a single output, y. This neural network is described as *fully-connected* because each unit is connected to every unit in the next layer. The "width" of the network refers to the number of units in each layer, while the "depth" of the network refers to the number of layers in the network.

The units of each layer can be represented as a vector, so the input layer can be represented by the vector $\vec{x}$. The values for the $\vec{x}$ vector come directly from the inputs. The values for the units in the first hidden layer come from Equation (2.16),

$$\vec{h} = g(W^T \vec{x} + \vec{b}) \tag{2.16}$$

which contains the previous layer's values, $\vec{x}$, a weighting matrix, W, a bias vector, $\vec{b}$, and a nonlinear activation function, g [25].

Each line connecting one node to another node in Figure 2.4 represents a weight, which appears in the weighting matrix W. The value $h_i^{(j)}$ represents the value of the i-th unit of the j-th hidden layer. To calculate the value for $h_1^{(1)}$, the first calculation is to take the sum of each input multiplied by its corresponding weight, and the next calculation is to add a bias term to this new sum. The new value is put into the activation function, g, such as the rectified linear unit (ReLU) function shown in Equation (2.17) [25]

$$g(x) = \max(0,\ x) \tag{2.17}$$

where $x = W^T\vec{x} + \vec{b}$. Although the ReLU function is one of the most commonly used activation functions, there are many other activation functions used for neural networks such as the sigmoid function and the hyperbolic tangent function, shown in Figure 2.5 [2].

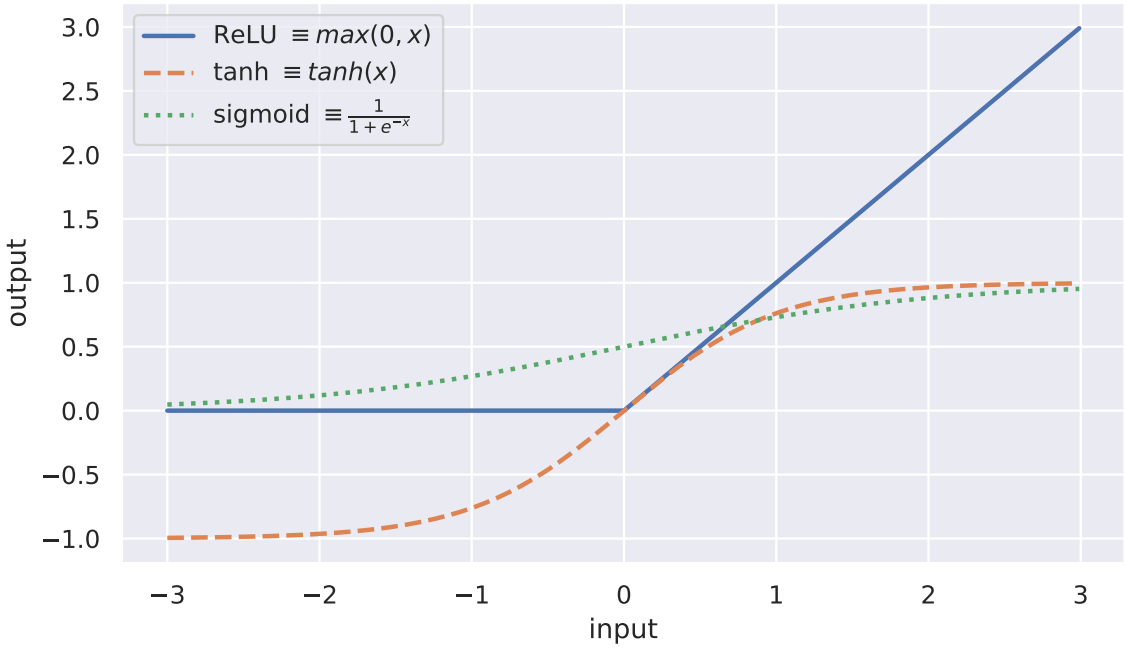

Figure 2.5. Common activation functions for neural networks

The output of this activation function is the value contained in the node $h_1^{(1)}$. Equation (2.16) shows the calculation for the entire layer rather than a single unit, which makes calculation much more efficient. Similar calculations are performed for the second hidden layer and for the output layer in this example.

Figure 2.4 shows a fully-connected feedforward neural network, which is one of the simplest architectures for neural networks. There is a large variety of other architectures, two of which are convolutional neural networks (CNNs), which work with grid-like data, and recurrent neural networks (RNNs), which work with sequential data [25]. Regardless of architecture, a neural network used in a *deep learning* model will have more than one layer, where the number of layers determines the *depth* of the model [25].

For the neural network to learn the best way to map the inputs to the output, it must adjust the weights and biases to appropriate values. Like many other machine learning algorithms, the method by which neural networks adjust their parameters is typically through some form of gradient descent [25].

2.3.3 Recurrent Neural Networks.

The network shown in Figure 2.4 is an example of a feedforward neural network, where the entirety of a sample is used as an input to the neural network and processed all at once [2]. When working with sequence data, it may be beneficial to process each element in the sequence in order and maintain a *state* containing information about what the network has seen so far [2]. This state is reset after processing each independent sequence. A neural network that contains state information, or memory, is called a recurrent neural network (RNN). Figure 2.6 shows the main idea behind RNNs, that is, a neural network with a loop.

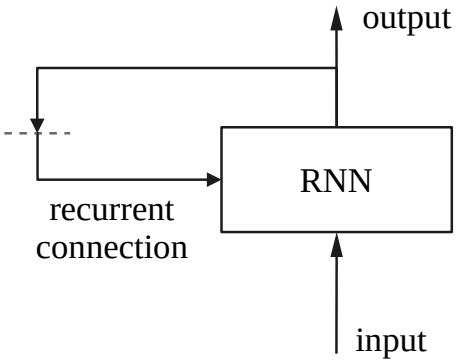

Figure 2.6. A simple recurrent network [2]

While RNNs may work well with sequence data, in their simplest forms, they often suffer from a vanishing or exploding gradient [27]. One remedy for this problem is to use the long short-term memory (LSTM) architecture, which can handle input sequences of up to 1,000 steps [27]. Figure 2.7 shows the anatomy of an LSTM network [2].

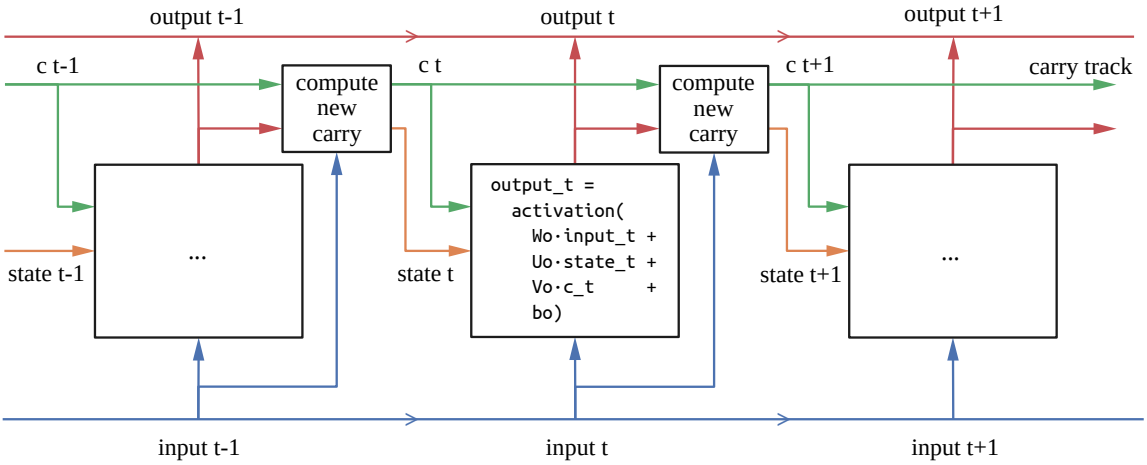

Figure 2.7. Anatomy of an LSTM [2]

At each timestep, the LSTM cell takes as input the current element of the sequence, the recurrent state, and information carried across timesteps, and the cell outputs a corresponding value [2]. The carry track is part of the difference between LSTMs and simple RNNs. The carry track allows past information to be processed

at a later step, which helps limit the vanishing gradient problem [2]. To compute the new carry, three transformations must be performed, each with their own weighting matrices, W and F, and bias vector, b, denoted by i, f, and k [2].

$$i_t = \text{activation}(U_i \cdot \text{state}_t + W_i \cdot \text{input}_t + b_i) \tag{2.18a}$$

$$f_t = \text{activation}(U_f \cdot \text{state}_t + W_f \cdot \text{input}_t + b_f) \tag{2.18b}$$

$$k_t = \text{activation}(U_k \cdot \text{state}_t + W_k \cdot \text{input}_t + b_k) \tag{2.18c}$$

$$c_{t+1} = i_t \cdot k_t + c_t \cdot f_t \tag{2.18d}$$

2.4 Reinforcement Learning

As shown in Figure 2.3, reinforcement learning (RL) can be considered another sub-domain of ML. While deep learning is a type of supervised learning where the dataset consists of inputs and labeled outputs, the data for reinforcement learning is generated as the agent interacts with the environment.

RL problems are typically formulated as Markov decision processes (MDPs) [3]. Figure 2.8 shows how an agent interacts with the environment in an MDP. The states in an MDP must have the Markov property, which means that the previous state contains all information necessary to transition to the next state [3]. For example, in the simple case of a ball falling to the Earth in a vacuum, the only state information needed to transition from one time step to the next would be the ball's position and velocity, assuming the acceleration due to gravity and the time step are defined in the environment. Figure 2.8 shows that the reinforcement learning agent chooses an action based on the current state and reward. This action is used to update the environment, which will update the state and reward for the next time step.

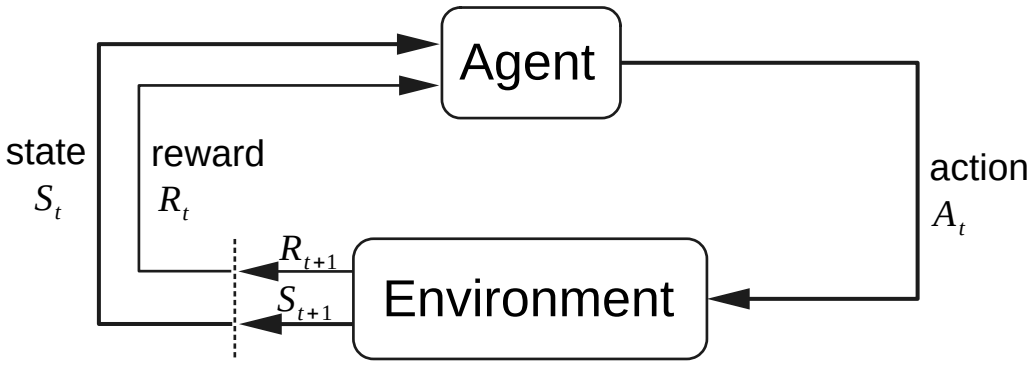

Figure 2.8. Agent-environment interaction in a Markov decision process [3]

The state, S_t, in an MDP is usually expressed as a vector or matrix, and contains all relevant information about the environment [4]. The reward signal, R_t, comes from the environment and is a measure of how "good" or "bad" the current state of the environment is [4]. The action, A_t, is the decision made by the agent that transitions the environment from one time step to the next [4]. The action is chosen which maximizes the reward, or typically the discounted future reward. The series of time steps until the terminal state is reached defines the length of an episode. An episode is a self-contained subsequence of agent-environment interaction, which could be a single game of chess, a trip through a maze, or any repeated interaction [3]. Each episode ends with a terminal state, and then resets to either a standard starting state or one drawn from a distribution of states [3]. Importantly, each episode begins independently of how the previous episode ended [3].

Two other fundamental components of a reinforcement learning agent are the policy and value functions, which are both found inside the "Agent" block in Figure 2.8. The policy is the set of rules used by the agent to determine which action to take next [4]. For example, the policy could be to take the same action at each time step, or it could be to sample the action from some normal distribution. The value function estimates the expected return starting from a given state and operating based on a

particular policy until the end of the episode, where the return is the sum of all future rewards (typically discounted) [4].

For problems with small enough state and action spaces, the value and policy functions can be stored in tables or arrays, and as such, the solutions for these types of problems are called tabular solution methods [3]. Tabular solution methods include dynamic programming, Monte Carlo methods, and temporal difference (TD) learning. These methods may even result in the optimal solution or policy [3].

For problems with larger state and action spaces, it may not be possible to store the value or policy functions as lookup tables. Instead, approximations for these functions are needed. While there are many ways to approximate these functions, one of the most commonly used function approximators is the neural network [3]. The combination of deep learning and reinforcement learning is called deep reinforcement learning.

2.4.1 Deep Reinforcement Learning.

Deep reinforcement learning takes advantage of the power of deep learning by using neural networks to approximate the policy and value functions. To give an idea of the types of deep reinforcement learning algorithms commonly in use, OpenAI's Spinning Up in Deep RL created a non-exhaustive taxonomy of RL algorithms shown in Figure 2.9 [4].

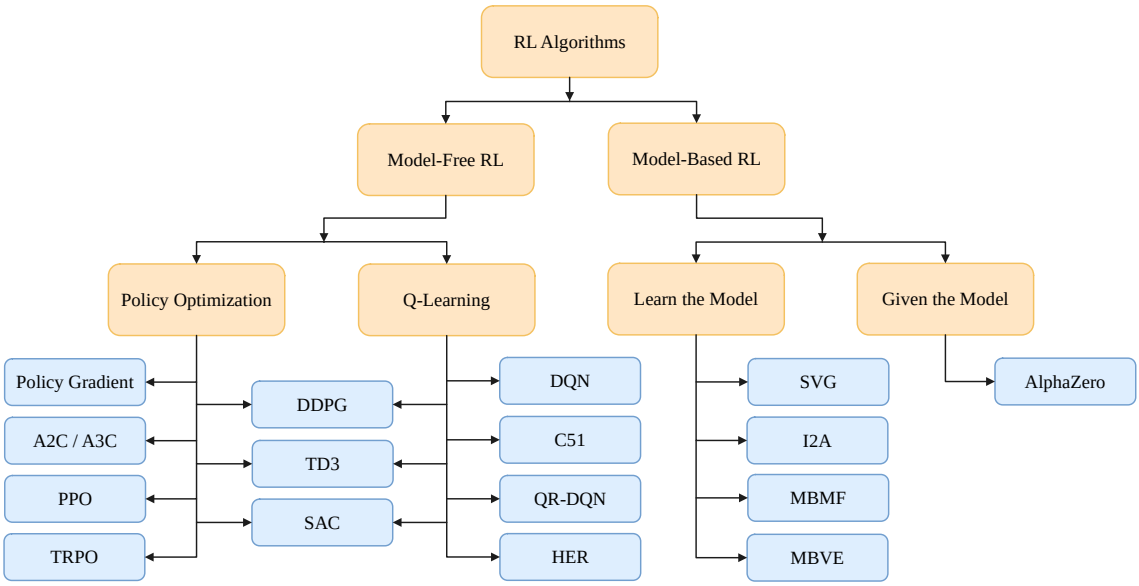

Figure 2.9. Taxonomy of reinforcement learning algorithms [4]

The first split in the RL taxonomy in Figure 2.9 is between model-free and model-based RL. The agent in model-based RL has access to the underlying model of the system, so the agent is better informed and may be able to plan ahead [4]. The agent in model-free RL has no knowledge of the underlying model, and these algorithms are typically easier to implement because they are not specific to a given problem [4]. Regardless of whether the algorithm is model-based or model-free, reinforcement learning algorithms can be very difficult to tune.

The next split for the model-free RL algorithms is between policy optimization and Q-learning. In policy optimization, the agent optimizes the parameters, θ, for the policy $\pi_\theta(a|s)$ by performing gradient ascent on the performance objective $J(\pi_\theta)$ or on a surrogate objective [4]. The policy, π, is parameterized by the values θ and returns an action, a, given the state, s. In Q-learning, the agent learns the action-value function, $Q_\theta(a, s)$, where the action, a, taken by the agent for any given state, s, will be the action with the largest Q value [4].

One common implementation for a variety of algorithms is called the actor-critic method. Actor-critic methods are those which learn to approximate both the policy and value functions [3]. The "actor" refers to the learned policy, and the "critic" refers to the learned value function [3]. Put more simply, the actor takes the action according to the policy, and the critic criticizes that action [3].

Another important distinction to make is between on-policy and off-policy algorithms. On-policy algorithms update the policy being used to take actions, whereas off-policy algorithms update a policy different from the one choosing the actions [3]. Figure 2.10 illustrates the difference between on-policy and off-policy algorithms and also shows how the actor-critic method is implemented for both.

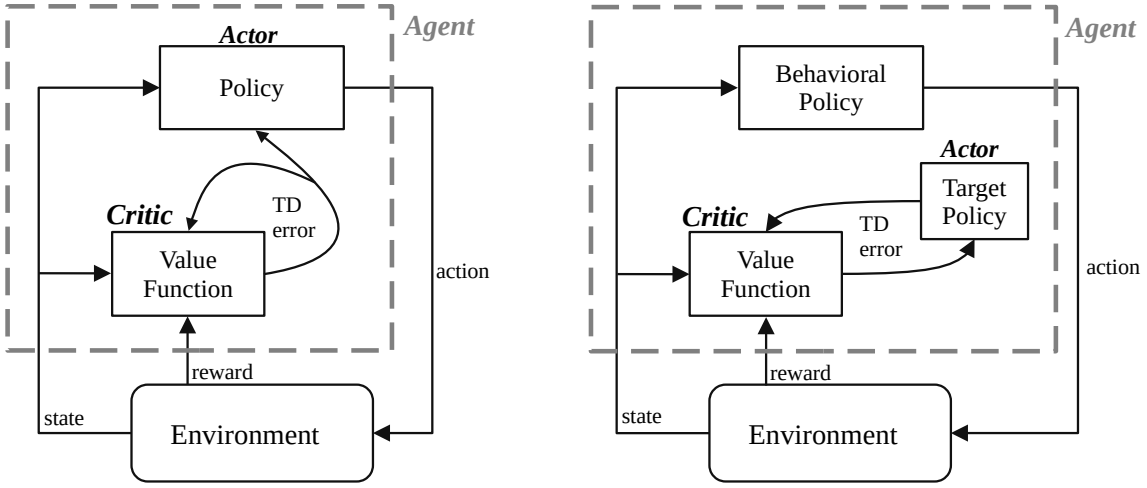

Figure 2.10. Example actor-critic architecture for on-policy (left) and off-policy (right) algorithms [3]

The left diagram in Figure 2.10 shows an on-policy algorithm. There is only one policy, and it is both choosing actions and being updated by the value function. The diagram on the right shows an off-policy algorithm where there are two policies: a behavioral policy and a target policy. The target policy is what the agent is learning and is being updated by the value function, while the behavioral policy is taking the actions and generating data for the agent to learn. One advantage of off-policy

learning is that it can be more sample efficient because it can reuse past experiences to speed up learning [4]. However, off-policy methods tend to be less stable and more brittle, meaning they are less likely to converge on a usable policy and are more sensitive to hyperparameters. While on-policy methods may not be as sample efficient, they do tend to be more stable, and less brittle [4].

2.4.2 Proximal Policy Optimization.

One popular and state-of-the-art deep reinforcement learning algorithm is called proximal policy optimization (PPO) [28]. This algorithm is model-free, on-policy, and works with continuous state and action spaces. PPO has been used successfully to win the world championships for a competitive video game called Dota 2 [29]. PPO has also been used to train an agent to solve a Rubik's cube in the real world with a robotic hand [30]. The PPO algorithm implemented in actor-critic style is shown in Algorithm 1.

Algorithm 1: PPO, Actor-Critic Style from [28]

for *iteration = 1, 2,...* **do**

  for *actor = 1, 2, ..., N* **do**

    Run policy $\pi_{\theta_{old}}$ in environment for T timesteps

    Compute advantage estimates $\hat{A}_1$, ..., $\hat{A}_T$

  end

  Minimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$

  $\theta_{old} \leftarrow \theta$

end

Like most deep reinforcement learning algorithms, PPO can run for as long as desired, and usually stops after a certain amount of time or steps, or after a certain performance threshold is met. Algorithm 1 begins with an infinite loop, but will terminate when one of the previously stated conditions are met. Like some other RL

algorithms, one benefit of PPO is that it can be run in parallel environments, with N actors. The next step in the algorithm is for each of the N actors to collect T timesteps of data from the environment using the policy parameterized by the old values, $\pi_{\theta_{old}}$. As each actor generates data from the environment, it will compute how much better the action is than expected, also known as the advantage, $\hat{A}$ [31]. After the $N \cdot T$ timesteps of data and advantage estimates are generated, the agent will optimize the surrogate loss function, L, for K epochs [28]. The new policy parameters then replace the old parameters, and the process repeats. Overall, the main idea behind the PPO algorithm is that there is a limit on how different new actions are compared to the old actions [28].

2.5 Reinforcement Learning Applications to Attitude Control

Artificial intelligence has been applied in many different fields with varying levels of success, and aerospace is no exception, with applications including planning, control, fault detection, and task allocation. Little and Frueh compare traditional algorithms, like the greedy algorithm, with machine learning algorithms, such as ant colony optimization and distributed Q-learning, with space situational awareness (SSA) sensor tasking [32]. Youmans and Lutze use supervised learning and linearized Clohessy-Wiltshire equations for both open-loop and closed-loop control of a satellite [33]. Coulter and Moncayo use support vector regression and principle component analysis, both machine learning techniques, for threat detection, isolation, and recovery for a spacecraft power system [34]. Chen and Cruz use a genetic algorithm, which is another field of AI, to allocate tasks to multiple unmanned aerial vehicles (UAVs) [35].

Another field of research in aerospace engineering is spacecraft MOI estimation. Hess uses a two-step optimization process exploiting the polhode curve to estimate

the MOI given the angular velocity measurements under the presence of stochastic measurement noise [36]. Keim et al. show that a constrained least squares approach to estimating spacecraft MOI can yield a more accurate solution than unconstrained least squares solutions [37]. Thienel et al. make use of a nonlinear passivity adaptive control scheme to estimate the six inertia parameters of a rigid spacecraft [38].

There have been many highly publicized examples of reinforcement learning being used to play games against humans like DeepMind's AlphaZero with Go and Chess [39], and OpenAI Five with Dota 2 [29]. These examples demonstrate the power of reinforcement learning, but they are not representative of all the problems that can make use of reinforcement learning. One application area is attitude control for both aircraft and spacecraft because of the complex and nonlinear dynamics involved.

Koch et al. use a variety of reinforcement learning algorithms to control the attitude of an UAV [40]. The authors compare PPO, trust region policy optimization (TRPO), and deep deterministic policy gradient (DDPG) algorithms, and also include a PID controller. They found that PPO overwhelmingly performs the best out of the algorithms tested for their specific scenario in simulation. Not only did the agent trained with the PPO algorithm converge to a solution more quickly, but it also converged to a higher average reward than agents trained with the other algorithms.

Focusing on just one algorithm, Bøhn et al. use the PPO algorithm for UAV attitude control, citing the work done by Koch et al. as the reasoning for choosing PPO [41]. Starting from a randomized initial condition, the RL agent is tasked with returning the UAV to a reference roll, pitch, and airspeed. The authors compare the RL agent to a PID controller and find similar results, but the RL agent appears to be more robust to turbulence disturbance in simulation.

Moving quite a bit higher in altitude, Wang et al. use the DDPG algorithm for autonomous spacecraft rendezvous guidance [42]. The authors formulate a planar,

2-degree-of-freedom (DOF) docking scenario, where a chaser spacecraft begins 10 km from the target spacecraft, and must reach a point 500 m from the target. This scenario does not include attitude dynamics, but the agent did achieve similar performance in the 2-DOF scenario to a PD controller.

Increasing the degrees of freedom by one, Hovell and Ulrich train a reinforcement learning agent to dock with a rotating target spacecraft in a 3-DOF scenario [43]. The authors used the distributed distributional deep deterministic policy gradient (D4PG), which is also an actor-critic algorithm. They use the term *Deep Guidance* to clarify that the RL agent is providing guidance commands that are tracked by a conventional controller. To test the agent during training, a proportional velocity controller is used. The agent outputs the desired velocity, feeds it into the controller, and the controller then outputs the control effort. The reward function is calculated by generating a reward field which is zero at the goal, and becomes more negative linearly as the distance increases. The reward for each timestep is then the difference in reward field between the current timestep and the previous timestep. This means that the reward will be positive if the agent moves towards the target, and negative if it moves away. The agents achieved promising results for docking with both a stationary target and a rotating target. The authors even experimentally validated the agent in the real world on two air-bearing spacecraft platforms.

The works cited above investigate attitude control and rendezvous maneuvers from the air all the way to Earth orbit. However, to the extent of the author's knowledge, the literature is lacking in RL based 3-DOF attitude control of a spacecraft. Both [40] and [41] investigate attitude control of aircraft, which operate under a different set of dynamics from spacecraft. The study in [42] does not have a spacecraft attitude component, and the study in [43] only considers 1-DOF for the spacecraft attitude.

Training an RL agent to control a spacecraft's attitude in a 3-DOF environment opens the possibility of easily controlling a pair of spacecraft in a post-docking scenario.

III. Methodology

Chapter III introduces the two problems investigated in this thesis in more detail and then lists the hardware and software components used in this research. This chapter also explains the methodology behind generating random MOI and random quaternions. Next, the methodology for estimating MOI using recurrent neural networks is explained. Finally, the process used for training a reinforcement learning agent to control the attitude of a spacecraft is explained.

3.1 Problems Overview

There are two main topics investigated in this thesis, each with their own set of problems, which will be referred to as Problem A and Problem B. Both problems make use of machine learning. Problem A uses a recurrent neural network to estimate spacecraft MOI. Problem B uses RL to control spacecraft attitude.

3.1.1 Problem A: Moment of Inertia Estimation via Recurrent Neural Networks.

Problem A will make use of an LSTM neural network to estimate spacecraft MOI given a time history of angular velocity data and two subproblems will have external torques. Problem A will involve two datasets: one with no torques, and one with a gravity-gradient torque and a magnetic torque.

The first subproblem, A1, will investigate estimating the principal moments of inertia and will include three scenarios. In the first scenario, the neural network will receive the angular velocity data, generated from simulating torque-free motion, and the neural network will output the predicted principal MOI. The second scenario will make use of the second dataset, and the neural network will be given angular

velocity, orbital position, and magnetic torque as inputs, and will predict the principal MOI. There is a gravity-gradient torque present in the second scenario, but it will be unknown to the neural network. The third scenario will also make use of the second dataset, but the only information the neural network will be given is the angular velocity data, meaning the torques and orbital position will be unknown.

The second subproblem, A2, will include the same scenarios as in subproblem A1, but instead of estimating the principal MOI, the neural network will be estimating the relative MOI ratios, or Smelt parameters.

3.1.2 Problem B: Deep Reinforcement Learning Applied to Spacecraft Attitude Control.

Problem B will use the PPO algorithm to control the attitude of a spacecraft and will include three distinct subproblems.

The first subproblem, B1, for Problem B will have the agent start from an arbitrary attitude and learn to achieve a particular attitude. During training the initial attitude and MOI will be randomized, but the desired orientation will remain the same across episodes, so the agent will only ever be able to point to the same attitude.

The second subproblem, B2, builds on the subproblem B1, and instead of achieving the same orientation each time, the agent will learn to achieve arbitrary attitudes. During training, the desired orientation is now also randomized for each episode, meaning the agent will be able to point to any desired orientation.

The final subproblem, B3, will consist of two parts: B3-1 and B3-2. For both parts, the goal is to have the agent experience a simulated collision resulting in a change in MOI and angular velocity. The agents in subproblem B3 are not newly trained. Instead, subproblem B3-1 uses the agent trained for subproblem B1, and subproblem B3-2 uses the agent trained for subproblem B2. Subproblem B3 is more

of a test of generality and robustness for Agent B1 and Agent B2. In subproblem B3-1, the MOI and angular velocity are randomized halfway through the episode. In subproblem B3-2, the MOI, angular velocity, and desired quaternion are randomized halfway through the episode.

3.2 Hardware and Software

This section, 3.2, details the hardware used to run the algorithms and train the models and agents for both Problem A, MOI estimation, and Problem B, RL attitude control. The same hardware was used for both problems. The processor used was an Intel Core i7-9850H at 2.60 GHz (6 cores, 12 threads). The computer used had 32 GB RAM, and an NVIDIA Quadro T2000 (4 GB GDDR5) graphics card.

For both problems, the operating system used was Ubuntu 20.04 LTS. For Problem A, the two datasets were generated using MATLAB 2020a [44]. The neural network setup and training, as well as the plotting of results was accomplished using Python 3.6 [45]. The Python library used for working with neural networks was Keras [46], which used the Tensorflow library as a backend [47].

Problem B was accomplished using Python 3.6 [45]. The reinforcement learning environment was created using the OpenAI Gym Python library [48]. Environment numerical integration was done using the SciPy Python library [49]. The reinforcement learning agent was created based on the Stable Baselines Python library [50].

3.3 Generating Random Samples

Because neural networks, and therefore deep reinforcement learning, require a large number of samples that span some desired range of operation, a method to

generate these samples must be implemented. This section explains methods for generating random moments of inertia and random quaternions.

3.3.1 Random Moments of Inertia.

The MOI of a rigid body describe how the mass of the body is distributed about the axes of a body-fixed reference frame [16]. A rigid body's products of inertia are a measure of how the mass is distributed about the three planes formed by the body-fixed reference frame [16]. The reference coordinate system can be oriented in such a way that all of the products of inertia are zero. In this orientation, the MOI and the reference frame axes are called the principal MOI and the principal axes respectively [16]. All physically realizable inertia matrices must obey the following constraints [51]:

- The inertia matrix J, when rotated by the matrix A via $A^T JA$, must have positive MOI, regardless of the matrix A [51]
- The inertia matrix must obey the triangle inequality principle, which states that the sum of any two MOI (principal or not) must be greater than or equal to the third MOI [51]
- The products of inertia are limited by the simultaneous application of both the above constraints [51]

In [51], Peck describes a process for generating physically realizable, random MOI. First, three numbers must be randomly sampled from a normal distribution with a mean of zero, μ, and a desired standard deviation, σ: $X \sim \mathcal{N}(0,\, \sigma^2)$. Each of the three randomly sampled numbers will be referred to as K_i. Each number will then be squared, so the resulting distribution of each K_i random variable is χ^2 because it is the square of a normal distribution [52]. Equation (3.1) produces the principal MOI,

ensuring the triangle inequality holds. The MOI can also be sorted from largest to smallest.

$$\begin{bmatrix} J_x \\ J_y \\ J_z \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} K_1^2 \\ K_2^2 \\ K_3^2 \end{bmatrix} \tag{3.1}$$

The resulting inertia matrix is constructed using Equation (3.2). Because this matrix represents the principal MOI, there are no products of inertia, which are the off-diagonal terms [19].

$$J = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix} \tag{3.2}$$

For simplicity, these steps are consolidated into the following algorithm:

Algorithm 2: Generate random moments of inertia [51]

---

1. $K_i \sim \mathcal{N}(\mu,\, \sigma^2),\ \ i = 1,\ 2,\ 3$

2. $$\begin{bmatrix} J_x \\ J_y \\ J_z \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} K_1^2 \\ K_2^2 \\ K_3^2 \end{bmatrix}$$

3. $$J = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix}$$

Using Algorithm 2 with $\mu = 0$ and $\sigma = 3$, 10,000 samples of MOI were randomly generated, and the histograms for each principal MOI are shown in Figure 3.1. The

histograms show that the vast majority of samples fall within a lower range, with a smaller number of samples having larger values of MOI.

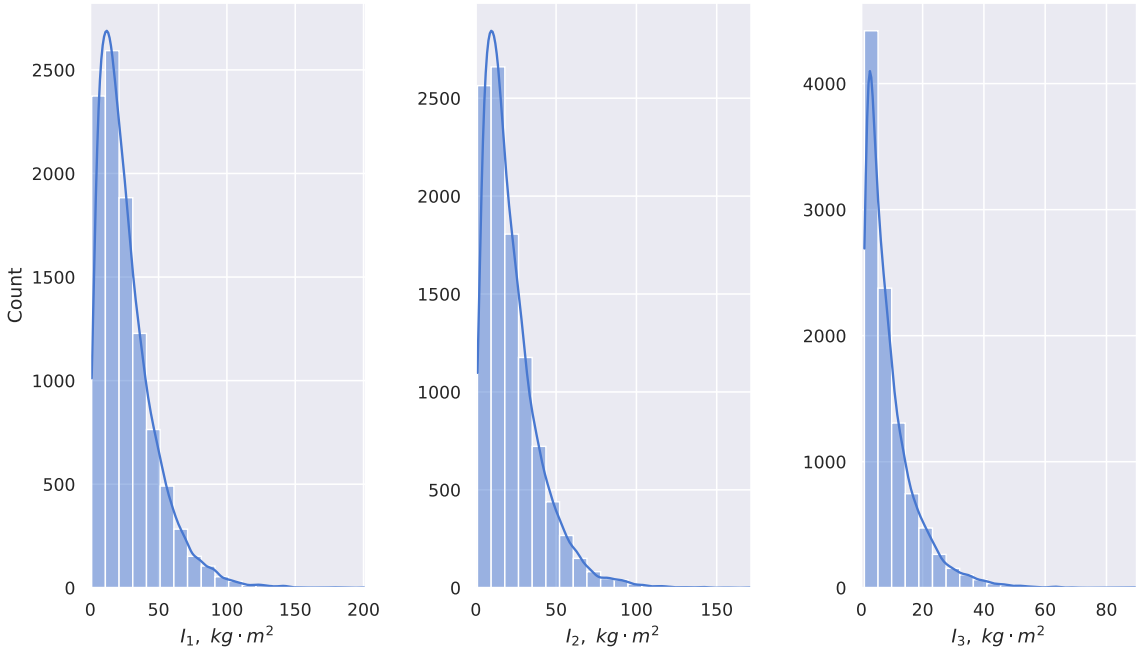

Figure 3.1. Distribution of randomly generated MOI

Sometimes, it is also useful to calculate the relative MOI ratios, which are known as the Smelt parameters [36]. The Smelt parameters are given by Equation (3.3) [53].

$$k_1 = \frac{J_y - J_z}{J_x} \tag{3.3a}$$

$$k_2 = \frac{J_z - J_x}{J_y} \tag{3.3b}$$

$$k_3 = \frac{J_x - J_y}{J_z} \tag{3.3c}$$

These parameters are not independent, but instead must satisfy the constraint in Equation (3.4) [53].

$$k_1 + k_2 + k_3 + k_1 k_2 k_3 = 0 \tag{3.4}$$

The result of this constraint is that all Smelt parameters must be found on the surface of the "petal" shape shown in Figure 3.2.

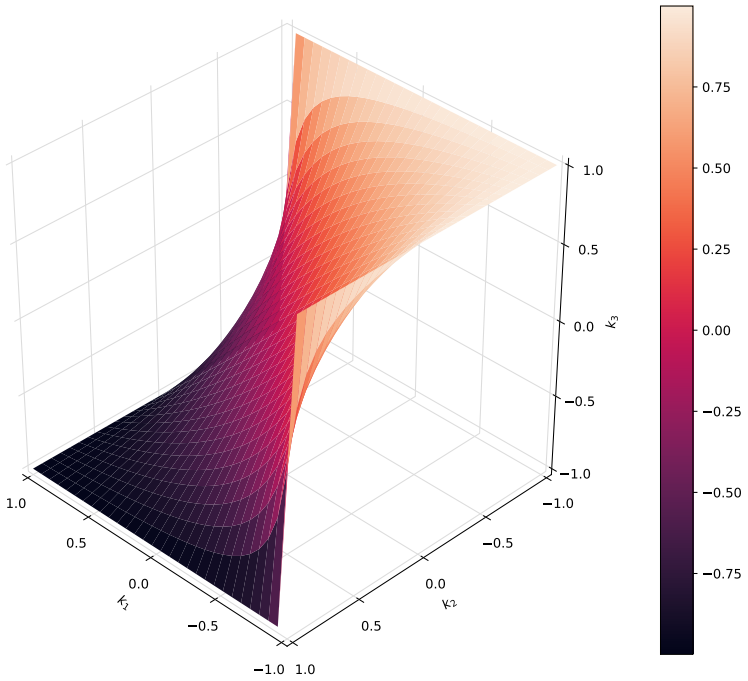

Figure 3.2. Smelt parameter constraint "petal"

When there are no torques acting on the spacecraft, Euler's equations of motion become those shown in Equation (3.5).

$$\dot{\omega}_x = k_1 \omega_y \omega_z \tag{3.5a}$$

$$\dot{\omega}_y = k_2 \omega_z \omega_x \tag{3.5b}$$

$$\dot{\omega}_z = k_3 \omega_x \omega_y \tag{3.5c}$$

3.3.2 Random Quaternions.

In [54], Shoemake describes a process for generating uniform random rotations. The first step is to sample three values, x_0, x_1, and x_2, from a standard uniform

distribution $X \sim U(0,\ 1)$. From these random variables, two uniformly distributed angles can be computed, which are shown in Equation (3.6).

$$\theta_1 = 2\pi x_1 \tag{3.6a}$$

$$\theta_2 = 2\pi x_2 \tag{3.6b}$$

The resulting quaternion, which maintains a unit norm, can be expressed as shown in Equation (3.7)

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \sin(\theta_1)\sqrt{1-x_0} \\ \cos(\theta_1)\sqrt{1-x_0} \\ \sin(\theta_2)\sqrt{x_0} \\ \cos(\theta_2)\sqrt{x_0} \end{bmatrix} \tag{3.7}$$

The steps for generating a random quaternion are consolidated into the following algorithm:

Algorithm 3: Generate random quaternion [54]

$x_i \sim U(0,\ 1), \quad i = 0,\ 1,\ 2$

$\theta_i = 2\pi x_i, \quad i = 1,\ 2$

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \sin(\theta_1)\sqrt{1-x_0} \\ \cos(\theta_1)\sqrt{1-x_0} \\ \sin(\theta_2)\sqrt{x_0} \\ \cos(\theta_2)\sqrt{x_0} \end{bmatrix}$$

*__if__ $q_4 < 0$ __then__
  $q = -q$

__end__

*Use if only shortest rotation desired

To ensure the randomly generated quaternions are indeed uniformly distributed, a large number of samples can be generated and converted to their corresponding Euler axis, $\hat{e}$, and Euler angle, α [18]. Equations (3.8) and (3.9) first calculate the Euler rotation angle, and then use that angle along with the vector component of the quaternion to calculate the Euler axis [20].

$$\alpha = 2\cos^{-1}(q_4) \tag{3.8}$$

$$\hat{e} = \frac{1}{\sin(\alpha/2)} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}; \quad \alpha \neq 0,\ \pm 2\pi \ldots \tag{3.9}$$

Singularities occur at $\alpha = 0,\ \pm 2\pi, \ldots,$ which correspond to orientations where $q_4 = 1$, or $q_4 = -1$. Because this conversion is used only for visualization purposes, orientations that cause a singularity are not included in Figure 3.3. With the equations for converting a quaternion to an Euler axis and angle, the Euler axis of each randomly generated quaternion can be plotted as a point in a scatter plot. Figure 3.3 shows Euler axes from 2,000 randomly generated quaternions plotted as points. The scatter plot on the left shows that the attitudes are uniformly distributed about the unit sphere. There is no apparent "bunching" around certain attitudes. The scatter plot on the right shows that specific sections of the unit sphere can be chosen by changing the range of values from which x_0, x_1, and x_2 in Algorithm 3 are chosen.

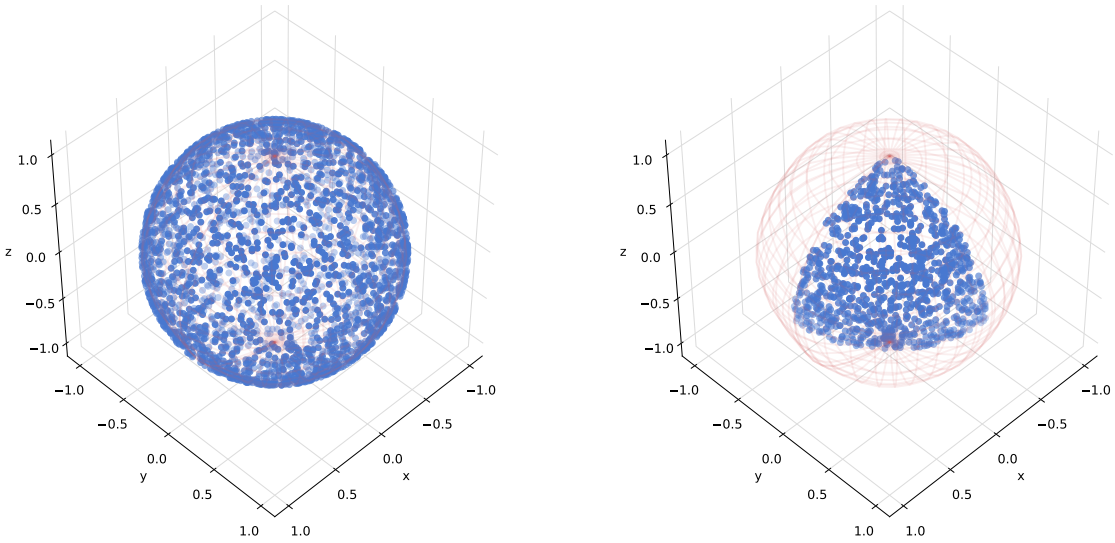

Figure 3.3. Distribution of 2000 randomly generated quaternions. The left figure shows the quaternions uniformly distributed about the entire unit sphere, while the figure on the right shows the samples concentrated on a smaller region of the unit sphere.

3.4 Problem A: Moment of Inertia Estimation via Recurrent Neural Networks

The work regarding MOI estimation shown in this section was presented at the 2020 AIAA / AAS Astrodynamics Specialist Conference [55]. For this problem, two datasets were generated using MATLAB and Simulink [44], each containing 25,000 samples. Each sample contains 30 seconds of data with 0.1 second timesteps, for a total of 300 steps per sample.

First, a random set of MOI is generated according to Algorithm 2, and the inertia values are sorted from largest to smallest. The initial angular velocities in each direction are set arbitrarily to $\vec{\omega}_0 = [0.2,\ 0.15,\ -0.18]^T \frac{rad}{s}$ for each sample. By keeping the initial angular velocities the same for each sample, the neural networks have a smaller space to search. If the neural network is unable to learn a useful mapping, when the initial angular velocity is always the same, then it would be unlikely to learn when the initial angular velocity is randomized. With the MOI and initial angular

velocity defined, Euler's equations of motion found in Equation (2.10) are used, with zero torque, to propagate the angular velocity for 30 seconds.

Whereas the first dataset contains only MOI and angular velocity with zero torques, the second dataset contains two external torques. The first is a gravity gradient torque, calculated using Equation (2.11). Because the gravity gradient torque includes the vector $\vec{R}_c$, which is the vector originating at the center of the Earth and pointing to the spacecraft center of mass, the orbit of the spacecraft must also be accounted for. The inertial position and velocity vectors are propagated using the two-body equation of motion shown in Equation (3.10) [19].

$$\ddot{\vec{R}}_c + \mu \frac{\vec{R}_c}{R_c^3} = 0 \tag{3.10}$$

The inertial position vector must then be transformed to the body frame by pre-multiplying the vector by a transformation matrix or DCM. For a more in-depth discussion on reference frames, see [20]. The second dataset also includes a commanded magnetic torque, calculated using Equations (2.15) and (2.12), where the gain for the "B-dot" steering law is set to $k = 100$.

3.4.1 Subproblem A1: Estimate Principal Moments of Inertia.

Three scenarios were developed based on the two datasets. The first scenario uses the first dataset, where the neural network is attempting to estimate the MOI given only the angular velocity data, with no external torques present. The second scenario uses the second dataset, and the neural network receives the angular velocity and the magnetic torques as inputs, and it attempts to estimate the MOI. The gravity gradient torque remains unknown to the neural network for Scenario Two. The third scenario also uses the second dataset, so there are external torques present, but the

neural network only has knowledge of the angular velocity, and both torques are unknown.

To estimate the MOI, a neural network is constructed consisting of five LSTM layers, each with 512 units. For each sample that the neural network sees, the output of the neural network is compared to the actual MOI, and the mean squared error (MSE) is calculated as the loss function. The neural network attempts to minimize the MSE between the estimated MOI and the actual MOI. The equation for MSE is shown in Equation (3.11), and represents the loss function, which the neural network attempts to minimize.

$$MSE_{loss} = \frac{1}{3}\left[(J_{x,est} - J_{x,truth})^2 + (J_{y,est} - J_{y,truth})^2 + (J_{z,est} - J_{z,truth})^2\right] \quad (3.11)$$

Equation (3.11) shows the loss function used by the neural network. However, when evaluating the results of the trained network, it may be more useful to calculate the mean squared error about each of the three axes separately. Once trained, if the neural network is given n samples to evaluate, then the equation used to calculate mean squared error is given by Equation (3.12). Alternative calculations are shown in Equations (3.13) and (3.14), which represent the mean absolute error (MAE) and the mean absolute percentage error (MAPE).

$$MSE = \frac{1}{n}\sum_{j=1}^{n}\begin{bmatrix}(J_{xn,est} - J_{xn,truth})^2\\(J_{yn,est} - J_{yn,truth})^2\\(J_{zn,est} - J_{zn,truth})^2\end{bmatrix} \quad (3.12)$$

$$MAE = \frac{1}{n}\sum_{j=1}^{n}\begin{bmatrix}|J_{xn,truth} - J_{xn,est}|\\|J_{yn,truth} - J_{yn,est}|\\|J_{zn,truth} - J_{zn,est}|\end{bmatrix} \quad (3.13)$$

$$MAPE = \frac{1}{n}\sum_{j=1}^{n} \begin{bmatrix} \left|\frac{J_{xn,truth}-J_{xn,est}}{J_{xn,truth}}\right| \\ \left|\frac{J_{yn,truth}-J_{yn,est}}{J_{yn,truth}}\right| \\ \left|\frac{J_{zn,truth}-J_{zn,est}}{J_{zn,truth}}\right| \end{bmatrix} \cdot 100 \tag{3.14}$$

3.4.2 Subproblem A2: Estimate Smelt Parameters.

Whereas subproblem A1 predicts the MOI for three different scenarios, subproblem A2 predicts the Smelt parameters for those same scenarios. The same datasets are used for subproblem A2, but instead of the neural network estimating MOI, it estimates the Smelt parameters, and compares the estimates to the actual Smelt parameters, which are calculated when loading the data. The neural network uses the same loss function of MSE as in subproblem A1. The architecture and hyperparameters of the neural network are the same for both subproblems A1 and A2 and for all three scenarios within each problem.

3.4.3 Architecture and Hyperparameters.

The neural network used for Problem A consists of five LSTM layers of 512 units each, and ends with a dense, or fully-connected, layer of 3 units, corresponding to the three outputs: MOI for subproblem A1, and Smelt parameters for subproblem A2. Figure (3.4) shows the neural network architecture for the first scenario in subproblem A1, where the other scenarios and subproblem A2 would have different inputs and outputs.

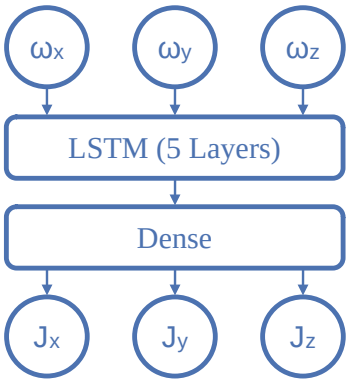

Figure 3.4. Neural network architecture for subproblem A1

There is no regularization because there were no signs of the neural network overfitting to the data. The network used a batch size of 128, and a learning rate of 0.001. The neural network uses the Adam optimizer, which is a stochastic gradient descent method based on adaptive estimation of first-order and second-order statistical moments [46]. The dataset is split into three parts: training, validation, and testing. The training dataset contains 18,000 samples, the validation set has 2,000 samples, and the test set has 5,000 samples.

3.5 Problem B: Spacecraft Attitude Control via Reinforcement Learning

Typical attitude control involves creating a model of the spacecraft being controlled, and deriving a control law for the spacecraft [19]. Some techniques, such as adaptive control and MPC, allow for some uncertainty in the system model, but they still require a model of the system. Many reinforcement learning techniques, on the other hand, are model-free and can learn a control policy with no knowledge of the underlying model of the system. Problem B applies model-free reinforcement learning to control the attitude of a spacecraft. The first subproblem involves starting from an arbitrary attitude, with an arbitrary MOI, but always ending in the same orientation. The second subproblem expands on the first part, and instead learns to achieve

any desired orientation. The final subproblem expands on the first two subproblems, testing the response of the agents to an abrupt change in MOI.

To accomplish these tasks, a few common open-source software packages are used. The first is OpenAI Gym, which is a toolkit for developing reinforcement learning environments [48]. OpenAI Gym allows the user to create a reinforcement learning environment, which defines observations, rewards, and how actions change the observations at each time step. The other piece of software is Stable Baselines, which is a set of RL algorithm implementations and is made to work with OpenAI Gym environments [50]. The Stable Baselines library has several algorithms available to choose as a starting point, which can then be modified to fit the user's needs.

3.5.1 Reinforcement Learning Environment.

This section details the OpenAI Gym environment created to solve Problem B. There are three functions that are required to create an environment: initialization, reset, and step. The reward function can either be its own separate function, or it can be calculated inside the step function.

3.5.1.1 Initialization.

This function defines what happens anytime a new instance of this environment is created. One of the first steps is to determine what the actions and observations will be, and define both the action and observations spaces, which will determine the maximum range of values the actions and observations can take. The actions and observations are defined in Equations (3.15) and (3.16) along with the maximum value for each.

$$\vec{a} = \begin{bmatrix} \hat{\omega}_x \\ \hat{\omega}_y \\ \hat{\omega}_z \end{bmatrix}; \qquad \vec{a}_{max} = \pm \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{3.15}$$

For this problem, the action corresponds to the requested angular velocity about each of the three axes. The "hat" character denotes that these angular velocity values are normalized and will always fall between $\pm$ 1. The reason the action is limited to be between $\pm$ 1 is to make it easier for the neural network to learn to map the action space. The actual maximum value of the requested angular velocity is $\omega_{max} = 0.1\ \frac{rad}{s}$, and the actions will be scaled by this value in the *step* function. Equation (3.16) represents the observation vector, $\vec{x}$, consisting of 17 components.

$$\vec{x} = \begin{bmatrix} q \\ \vec{\omega} \\ \vec{\tau} \\ q_d \\ \vec{\omega}_d \end{bmatrix} \tag{3.16}$$

The first seven components of the observation vector in Equation (3.16) correspond to the spacecraft's attitude, q, and its angular velocity, $\vec{\omega}$. The next three components correspond to the requested torque, $\vec{\tau}$, and although the agent does not require knowledge of the torque at any time, it makes plotting the control history much easier. The final seven components correspond to the desired attitude, q_d, and the desired angular velocity, $\vec{\omega}_d$, for the spacecraft. These values are included so that the agent knows where it should be pointing. It gives the agent more direct feedback than just a reward function would, and it allows the desired attitude and angular velocity to be updated more easily. The maximum values for q and q_d are both ± 1 because they are both unit quaternions. The maximum values for $\vec{\omega}$ and $\vec{\omega}_d$ were chosen to be $\pm 100\ \frac{rad}{s}$ and $\pm 100\ Nm$ because these values are well outside the arbitrarily chosen range of normal operations.

The remainder of the initialization process involves setting the initial values for the rest of the variables. The time step is set to `dt`= 0.2 seconds. The maximum

number of steps for each episode is set to `max_steps` $= 1000/dt$. The MOI values are all set to 10 $kg \cdot m^2$, but they will be randomized at the start of each episode. The average MOI is calculated by adding the three MOI values and dividing by three, and this average value will be used later to determine the maximum allowable requested angular velocity. A random number generator must be initialized. All 17 states are set to zero. The current step in the environment is set to $t = 0$. The Boolean value specifying whether the episode is over is set to `done` $= False$, and the current reward is set to zero.

3.5.1.2 Reset.

The next function for the Gym environment that must be defined is the *reset* function, which determines what happens to the environment when it is reset for the next episode. First, a random quaternion is generated using Algorithm 3 and the initial attitude is set to this new quaternion. Next, the initial angular velocities are randomly sampled from a normal distribution parameterized by $\mu = 0$ and $\sigma = 10\frac{deg}{s}$. The initial torques are all set to zero because no control has been requested yet. Next, a new set of MOI are generated using Algorithm 2 parameterized by $\mu = 0$ and $\sigma = 3$. The average MOI is then calculated, and the maximum allowed requested angular velocity in each axis is set to $\omega_{max} =$ `avg_moi`$/100$. The desired quaternion will depend on which problem is being solved. For subproblems B1 and B3-1, this value does not change over the course of all training and testing. For subproblems B2 and B3-2, the desired quaternion is randomized for each episode according to Algorithm 3. Finally, the timestep and reward are set to zero, and the `done` flag is set to $False$.

3.5.1.3 Reward Function.

The methodology behind the reward function used in this environment is a modified version of the reward function used by Hovell and Ulrich [43]. First, a field of values is defined that is zero at the goal, and becomes more negative further from the goal. This field of values is called the reward field. To calculate the reward field, the error quaternion between the current quaternion and the desired quaternion is calculated using Equation (2.6). There are then four values that must be calculated. The first value, r_0, is the negative sum of the absolute values of the vector components of the error quaternion, scaled by some value, α.

$$r_0 = -\alpha(|q_{e1}| + |q_{e2}| + |q_{e3}|) \tag{3.17}$$

The next three values are the negative absolute values of the difference between the desired angular velocity and the current angular velocity, scaled by some value, β.

$$\vec{r}_{1:3} = -\beta \begin{bmatrix} |\omega_{d,x} - \omega_x| \\ |\omega_{d,y} - \omega_y| \\ |\omega_{d,z} - \omega_z| \end{bmatrix} \tag{3.18}$$

The value of the reward field, r_f, is then the sum of each component.

$$r_f = r_0 + r_1 + r_2 + r_3 \tag{3.19}$$

This process showed the calculation of the reward *field*, but the value of the reward *function* will be calculated in the *step* function. In this problem, the reward field is simply an intermediate value used for determining the actual reward at any given time step.

3.5.1.4 Step.

The next function that must be defined is the *step* function, which defines how the agent steps through the environment at each timestep. This function takes an action as the input, which in this case is the requested angular velocity from the RL agent. First, the requested angular velocity is fed into a simple proportional controller of the form shown in Equation (3.20)

$$\vec{u} = K_p(\vec{a} \cdot \vec{w}_{max} - \vec{\omega}) \tag{3.20}$$

where $\vec{u}$ is the requested control, the proportional gain is notionally set to K_p= 2 for this study, the normalized action is $\vec{a}$, the maximum angular velocity is $\vec{w}_{max}$, and the current angular velocity is $\vec{\omega}$. Next, the sum of the control effort is calculated using the 1-norm.

$$u = \sum_{i=1}^{3} |u_i| = |u_1| + |u_2| + |u_3| \tag{3.21}$$

The next step is to calculate the reward reward field before stepping through the environment. This value will be called r^-, and it will be calculated using scaling values of $\alpha = 3$, and $\beta = 1$. Once this value is calculated, the environment will be updated using the requested control. The only states that must be updated are the four quaternion values, q, and the three angular velocity values, $\vec{\omega}$. However, to update the states, the control effort, $\vec{u}$, is also required. The system of differential equations used for updating the states is shown in Equation (3.22).

$$\vec{y} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}; \qquad \dot{\vec{y}} = \begin{bmatrix} 0.5(q_4\omega_x - q_3\omega_y + q_2\omega_z) \\ 0.5(q_3\omega_x + q_4\omega_y - q_1\omega_z) \\ 0.5(-q_2\omega_x + q_1\omega_y + q_4\omega_z) \\ 0.5(-q_1\omega_x - q_2\omega_y - q_3\omega_z) \\ \frac{1}{J_x}(u_1 + (J_z - J_y)\omega_y\omega_z) \\ \frac{1}{J_y}(u_2 + (J_x - J_z)\omega_z\omega_x) \\ \frac{1}{J_z}(u_3 + (J_y - J_x)\omega_x\omega_y) \end{bmatrix} \tag{3.22}$$

The timestep of the environment is 0.2 seconds, which is long enough to have the potential of discretization errors building up if the dynamics are not integrated at a high enough frequency. To fix this issue, the SciPy library was used [49]. The `solve_ivp` function was used with a relative tolerance of 10^{-3}, an absolute tolerance of 10^{-6}, and with the default RK45 solver [56]. In essence, the solver takes the current state as the initial condition, and propagates the trajectory forward in time by 0.2 seconds, where the control is constant over that time. The new state is then the final set of values from the solver. With the new state calculated, the new reward field, r_f^+, can also be calculated. A penalty, p_c, which is proportional to the sum of the control, u, and inversely proportional to the average MOI, J_{avg}, will be added to the final reward. The average MOI will always be $J_{avg} \geq 1$ because each MOI is kept to be $J \geq 1$.

$$p_c = 0.5u\frac{1}{J_{avg}} \tag{3.23}$$

The final reward can be calculated as shown in Equation (3.24)

$$r = r_f^+ - r_f^- - p_c \tag{3.24}$$

After the reward is calculated, the current step number of the environment must be incremented by one. If the current step is greater than or equal to the max number

of steps, then the Boolean flag, `done`, must be set to `True`. Finally, the *step* function returns the current state, the reward, and the `done` flag.

3.5.2 Hyperparameters.

The other main component of the solution for Problem B is the RL algorithm, which will be choosing actions and interacting with the environment. The RL agent is made using the PPO algorithm from Algorithm 1. Because the PPO algorithm calls for an actor-critic network, two neural networks will be created, although it is possible to have one network with two outputs. The actor, or policy network, will have three hidden layers of 16 units each. The critic, or value function network, will also have three layers of 16 units. Both networks are multi-layer perceptron networks, which is another way of saying fully-connected, feedforward networks. The actor-critic network is defined as a custom policy in the baseline library.

The Gym environment is vectorized so that the PPO agent can interact with four environments at the same time, and collect data in minibatches from all four environments. Most of the hyperparameters were left as the default values [50]. However, the learning rate was manually tuned until it yielded promising results. The hyperparameters are summarized in Table 3.1.

Table 3.1. Problem B Hyperparameters

Description	Value
Number of environments	`num_envs` $= 4$
Discount factor	`gamma` $= 0.99$
Number of steps per update	`n_steps` $= 64$
Entropy coefficient	`ent_coef` $= 0.01$
Learning rate	`learning_rate` $= 8 \times 10^{-5}, \ \ 6 \times 10^{-5}$
Value function coefficient	`vf_coef` $= 0.5$
Max value for gradient clipping	`max_grad_norm` $= 0.05$
Factor for bias vs variance trade-off	`lam` $= 0.95$
Number of minibatches	`nminibatches` $= 4$
Number of epochs when optimizing the surrogate	`noptepochs` $= 4$
Clipping parameter	`cliprange` $= 0.2$

3.5.3 Subproblem B1: Start from Arbitrary Attitude.

The goal of subproblem B1 is to have the agent start from an arbitrary attitude, with a randomized MOI, and achieve a specific orientation. This is the most basic of the three problems and serves as a proof of concept. If the agent cannot learn to achieve the same orientation each time, it is unlikely the agent would be able to achieve an arbitrary orientation. The initial orientation for each episode is generated according to Algorithm 3, and the MOI for each episode is randomized according to Algorithm 2. Equation (3.25) shows the desired quaternion for every episode in subproblem B1.

$$q_d = [0, \ 0, \ 0, \ 1]^T \tag{3.25}$$

Deep reinforcement learning can be difficult in part because of the large number of parameters that can be tuned. Not only are there hyperparameters for the reinforcement learning algorithm, such as learning rate and discount factor, but there are also a number of parameters for the environment, such as the reward function, and number of time steps. The process of creating a useful agent becomes a game of tuning hyperparameters. Apart from lowering the learning rate from the default value, most of the tuning was done with the environment parameters, specifically the reward function.

There is a trade off between penalizing quaternion error and angular velocity error. If the quaternion error is penalized too much compared to the angular velocity, then the agent will tend to oscillate around the desired quaternion. If the quaternion error is penalized too little compared to the angular velocity, then the agent will stop rotating as soon as possible, regardless of the orientation it ends in. While these two values were being adjusted, with no penalty on the control effort, the agent used as much as needed, which was often noisy and physically impossible. This means that on top of balancing quaternion error and angular velocity penalties, the control effort must also be penalized, which complicates the tuning of the other two penalties. If the control effort penalty is too high, the agent will either take too much time to achieve the desired attitude, or not move at all, and if too low, it will use too much control.

Once the reward function values were tuned adequately, the agent trained for 5 million time steps. Each episode goes for 2,000 seconds with a time step of 0.2 seconds, thus each episode is 10,000 timesteps. Alternatively, the agent trained for 500 episodes. On the hardware used in this thesis, Agent B1 took 51 minutes to train.

3.5.4 Subproblem B2: Achieve Arbitrary Attitude.

Subproblem B2 builds on subproblem B1, the only difference being that instead of always ending at the same orientation, the desired orientation is randomized at the start of each episode. This is a much more difficult problem for the agent to learn a solution for because the search space is much larger. Instead of starting from an infinite number of possible orientations and ending in one orientation, the agent must now start from an infinite number of possible orientations and end in an infinite number of possible orientations.

Agent B2 decreased the learning rate to 6.0×10^{-5}, from the previous value of 8.0×10^{-5} used in subproblem B1. A lower learning rate was chosen because it appeared to help the agent converge on a policy. The scaling factor on the control penalty was increased from 0.5 to 0.6 to mitigate the large control effort observed from previous iterations of Agent B2.

Subproblem B2 is a much more general problem than B1, which results in a much longer training time. The agent in subproblem B2 trains for 20 million steps, with an episode length of 1,500 seconds, resulting in 2,667 episodes. On the hardware used for this thesis, Agent B2 took 3 hours and 23 minutes to train. The episode length is one of the parameters that was tuned for this research. With a longer episode length, the agent accumulated more control penalty, and seemed to respond slower after training. By decreasing the episode length from 2,000 to 1,500 seconds, the agent seemed to achieve a greater balance between control effort and response time.

3.5.5 Subproblem B3: Simulated Collision.

Subproblem B3 builds on subproblems B1 and B2 by including a change in MOI. Subproblem B3 is split into two parts: B3-1 and B3-2. Subproblem B3-1 uses the agent from subproblem B1, and subproblem B3-2 uses the agent from B2. In both

cases, no new training takes place. Instead, the existing agents are evaluated in scenarios where the MOI changes.

For subproblem B3-1, the MOI is randomized at the middle of the episode. So for the 120 second episode, the MOI is randomized at 0 seconds and at 60 seconds. However, a change in MOI alone may not change the trajectory if the agent is already at rest. To ensure sufficient excitation in the response, the angular velocity is also randomized according to a normal distribution parameterized by $\mu = 0$ and $\sigma = 10\frac{deg}{s}$. The combination of a change in MOI and a change in angular velocity could simulate an external object making contact with and sticking to the spacecraft or a loss of structural components. The goal of the agent remains unchanged, as it is still attempting to achieve the desired quaternion from Equation (3.25).

Subproblem B3-2 is similar to B3-1, but it uses the agent trained for subproblem B2, which learned to achieve arbitrary orientations. Because Agent B2 can take longer to achieve orientations than Agent B1, the episode time was increase to 2,000 seconds, and the disturbances were introduced at 1,000 seconds. In addition to the change in MOI and change in angular velocity, subproblem B3-2 also changes the desired quaternion at the same time.

IV. Results and Analysis

This section presents and discusses the results of Problems A and B. The goal of Problem A is to predict the MOI and Smelt parameters of a spacecraft using a recurrent neural network. The goal of Problem B is to control the attitude of a spacecraft using the proximal policy optimization reinforcement learning algorithm.

4.1 Problem A

The goal of Problem A is to predict the MOI and Smelt parameters of a spacecraft given some combination of angular velocity and torque data using a recurrent neural network. The results of these predictions are shown in both tabular and graphical formats. The tables show the median MSE, MAE, and MAPE between truth and predicted MOI and Smelt parameters over all test samples about each axis.

One of the graphical representations of the results is in the form of three scatter plots, showing MSE, MAE, and MAPE, respectively. On the x-axis is the value of the inertia, and on the y-axis is the corresponding error value. The MOI values are split into their own categories and plotted separately.

The other graphical representation is in the form of three histograms, each representing J_x, J_y, and J_z, respectively. On the x-axis is the value of inertia, and on the y-axis is the density, representing how often that value appears in the data. Each plot has two histograms, one representing the distribution of the actual MOI and one representing the predicted MOI.

4.1.1 Subproblem A1: Estimate Principal Moments of Inertia.

There were 5,000 samples used as the test set, and Table 4.1 shows the median MSE, MAE, and MAPE values across all of the test samples about each axis.

Table 4.1. Subproblem A1 results: median MOI prediction errors

	MSE $(kg \cdot m^2)^2$			**MAE** $(kg \cdot m^2)$			**MAPE** (percent)		
	J_x	J_y	J_z	J_x	J_y	J_z	J_x	J_y	J_z
Scenario 1	183.78	115.35	11.92	13.56	10.74	3.45	54.92	54.65	58.99
Scenario 2	0.58	0.61	0.22	0.76	0.78	0.47	4.10	5.26	9.82
Scenario 3	9.98	6.91	0.66	3.16	2.63	0.81	14.90	15.27	16.39

The median values were chosen instead of the mean values because of the outliers, or large error values, that would skew the results. Table 4.1 shows that Scenario 1 had, by far, the largest errors of the three scenarios. This is expected because of the ambiguity present during torque-free motion. Under torque-free motion, the dynamics can be expressed using the Smelt parameters as shown in Equation (3.5). This means that it is not the absolute MOI that determines the dynamics, but rather the relative MOI ratios. For any given set of Smelt parameters, there are an infinite number of combinations of absolute MOI that can result in those Smelt parameters, when considering torque-free motion [36]. Because it is mathematically impossible to estimate absolute MOI from torque-free motion, Scenario 1 makes a good upper bound on the error values.

As expected, when there is a torque present, and the neural network has at least partial knowledge of this torque, as in Scenario 2, the errors are much smaller, suggesting that the neural network is able to predict the MOI. The errors in Scenario 3 fall somewhere between those of Scenario 1 and Scenario 2, although there are much closer to the errors in Scenario 2. This suggests that the neural network did learn a useful mapping between angular velocity and MOI, but it is not as accurate with-

out knowledge of the torque. To gain more insight into the results, the errors are presented in graphical format.

Figure 4.1 shows the results of Scenario 1 as three scatter plots each with logarithmic x and y axes. The scatter plots in Figure 4.1 show that as the value of the inertia increases, the error also increases. At a certain point, close to where the inertia is 20 kg·m^2, the absolute error between the estimated inertia and the truth inertia is almost exactly equal to the truth inertia, indicated by the almost straight line after this point. This observation, coupled with the large number of smaller errors at the same point suggests that the neural network is always predicting that the MOI is around 20 kg·m^2.

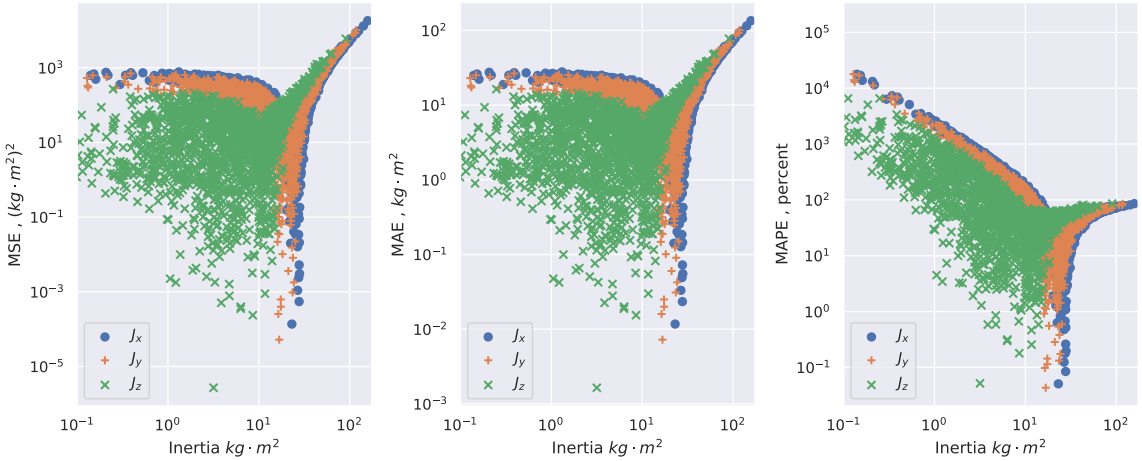

Figure 4.1. Subproblem A1, scenario 1 results

Figure 4.2 shows a histogram of the predicted distribution of inertia for Scenario 1 and the actual distribution of the inertia. These histograms confirm that the neural network is only predicting a small range of values instead of the wider range found in the actual distribution. The predicted distributions also appear to be bimodal, made apparent by the two peaks in each histogram. The actual distribution should match the MOI distribution shown in Figure 3.1.

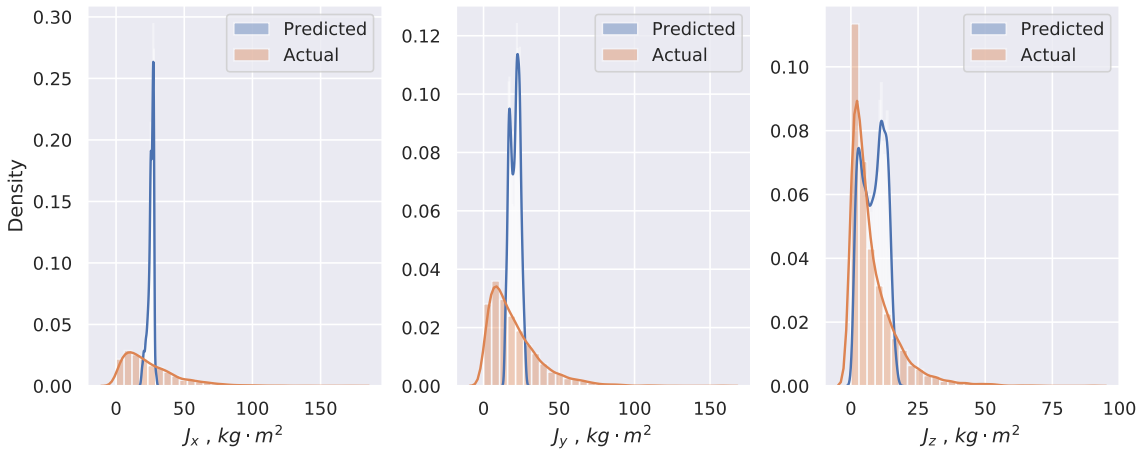

Figure 4.2. Subproblem A1, scenario 1 prediction distribution

Figure 4.3 shows that the errors for Scenario 2 are consistently smaller than for Scenario 1. However, the Scenario 2 errors also appear to increase as the value of the inertia increases. Unlike in Scenario 1, the errors for Scenario 2 seem more spread out.

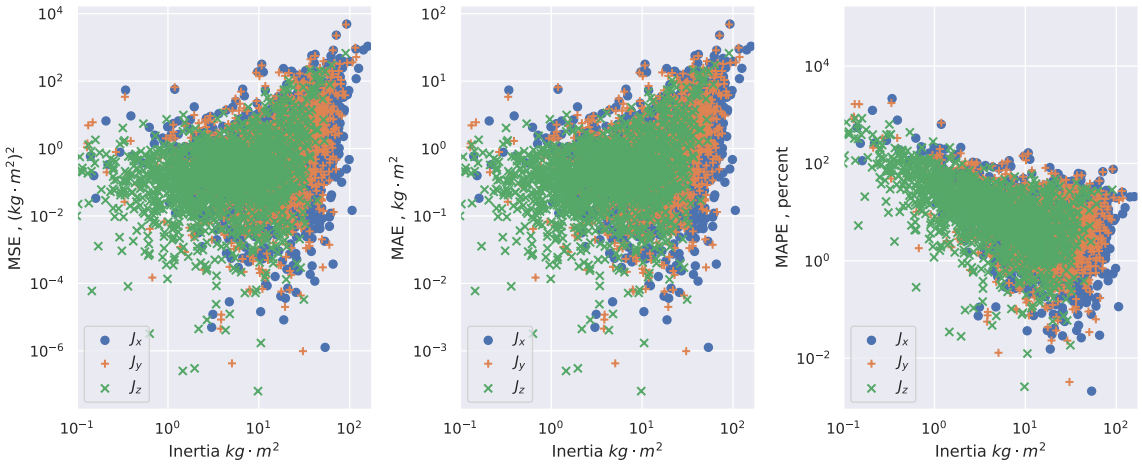

Figure 4.3. Subproblem A1, scenario 2 results

Figure 4.4 shows that the predicted distribution for Scenario 2 almost exactly matches the actual distribution. The matching of distributions suggests that the neural network learned a useful mapping from inputs to MOI for Scenario 2.

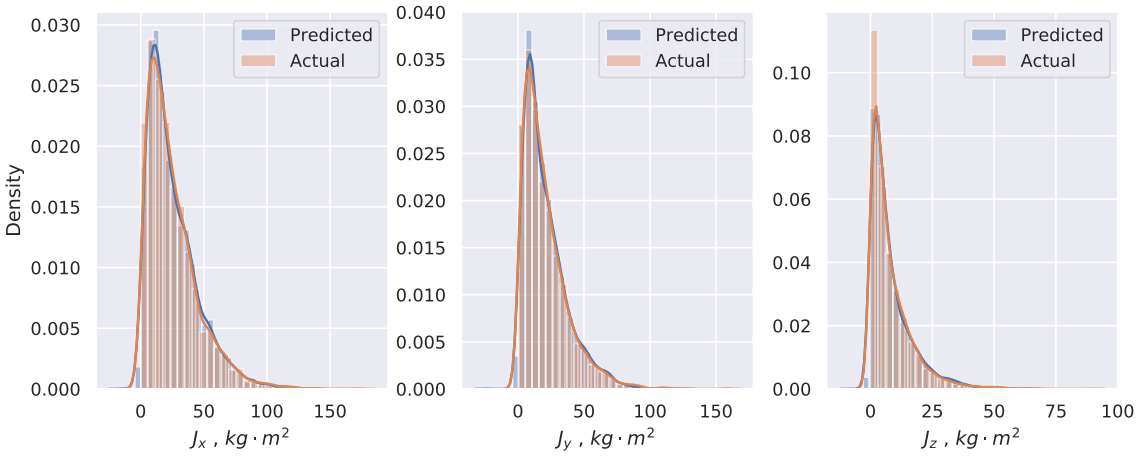

Figure 4.4. Subproblem A1, scenario 2 prediction distribution

Figure 4.5 shows that the errors for Scenario 3 are similar to those found in Scenario 2, but the Scenario 3 errors appear to have a larger increase in variance as the inertia values increase. The errors also appear to be larger than in Scenario 2. This suggests that the neural network for Scenario 3 learned a useful mapping, but not quite as useful as the Scenario 2 neural network. While not as prominent as in Scenario 1, the errors for Scenario 3 also appear to show some structure towards the larger MOI.

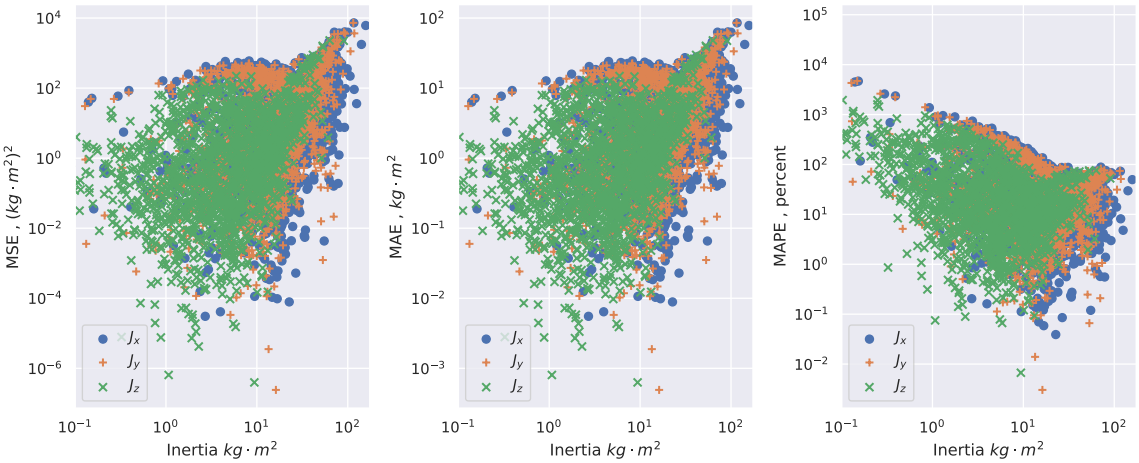

Figure 4.5. Subproblem A1, scenario 3 results

Figure 4.6 shows the predicted and actual MOI distributions for Scenario 3. The distributions appear similar, but not as similar as in Scenario 2. There also appears to be multiple modes, or peaks, just as in Scenario 1. It is unclear whether the neural network in Scenario 3 would converge to a similar level of error as Scenario 2 if there is more data or if the hyperparameters or network architecture are adjusted.

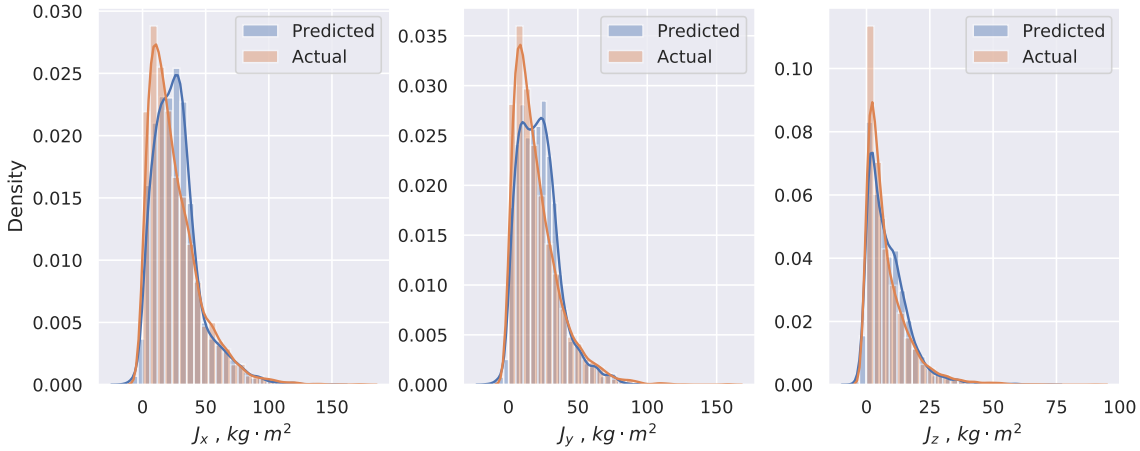

Figure 4.6. Subproblem A1, scenario 3 prediction distribution

4.1.2 Subproblem A2: Estimate Smelt Parameters.

The results for subproblem A2 are presented in the same format as in Problem A1, the only difference being that Smelt parameters are predicted instead of absolute MOI. Table 4.2 shows that all three scenarios have similar error values when predicting Smelt parameters, but that Scenario 1 has the lowest error values across all measures. This finding is the opposite of subproblem A1, potentially because there are fewer features for which the neural network must account. Scenarios 2 and 3 yielded similar errors suggesting knowledge of the torque is not required to predict Smelt parameters.

Table 4.2. Subproblem A2 results: median Smelt parameter prediction errors

	MSE (unitless)			MAE (unitless)			MAPE (percent)		
	k_1	k_2	k_3	k_1	k_2	k_3	k_1	k_2	k_3
Scenario 1	3.4×10^{-7}	1.9×10^{-7}	3.9×10^{-7}	0.00058	0.00044	0.00062	0.13	0.054	0.11
Scenario 2	8.1×10^{-7}	3.6×10^{-7}	1.2×10^{-6}	0.00090	0.00060	0.0011	0.18	0.75	0.22
Scenario 3	9.1×10^{-7}	5.7×10^{-7}	7.9×10^{-7}	0.00095	0.00076	0.00089	0.24	0.087	0.13

Figure 4.7 shows the scatter plots of the errors for Scenario 1. The difference between the predicted and actual Smelt parameters is very small for most samples, but just as for subproblem A1, it appears that the average error increases as the inertia values increase. This is most likely because of the sparse number of samples with large MOI, making it difficult for the neural network to learn how best work with these inputs.

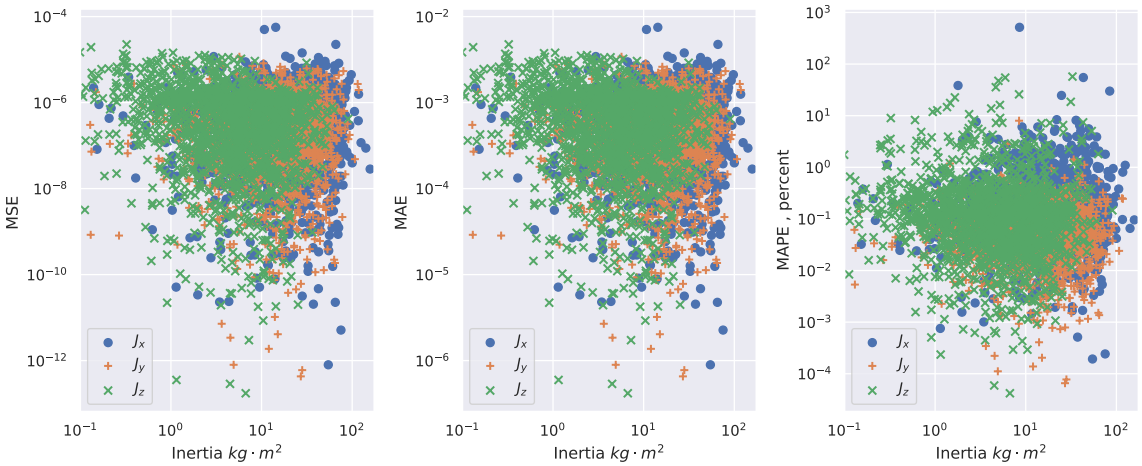

Figure 4.7. Subproblem A2, scenario 1 results

Figure 4.8 tells a similar story for Scenario 1. The distributions for Smelt parameters align almost perfectly. The distributions are so similar that it is difficult to distinguish the predicted from the actual distributions in Figure 4.8. The MOI are sorted such that $J_x \geq J_y \geq J_z$, which means that the first and third Smelt pa-

rameters, k_1 and k_3, are always positive, and the second, k_2, is always negative. The shapes of the distributions in Figure 4.8 are a result of the sorting of MOI and the Smelt parameter constraint from Equation (3.4).

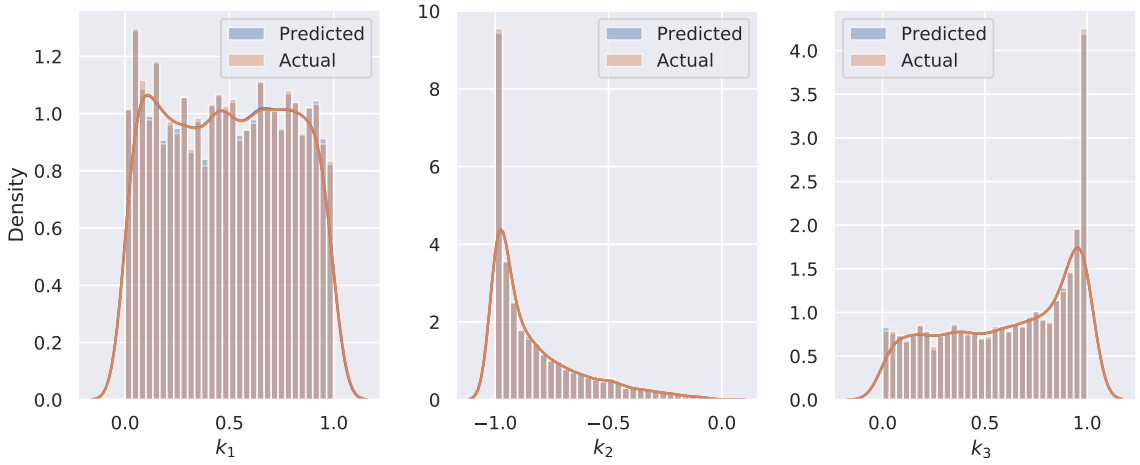

Figure 4.8. Subproblem A2, scenario 1 prediction distribution

Figure 4.9 shows that the errors for Scenario 2 are also very small, but not quite as small as in Scenario 1. The errors also appear to be larger when the inertia values are smaller.

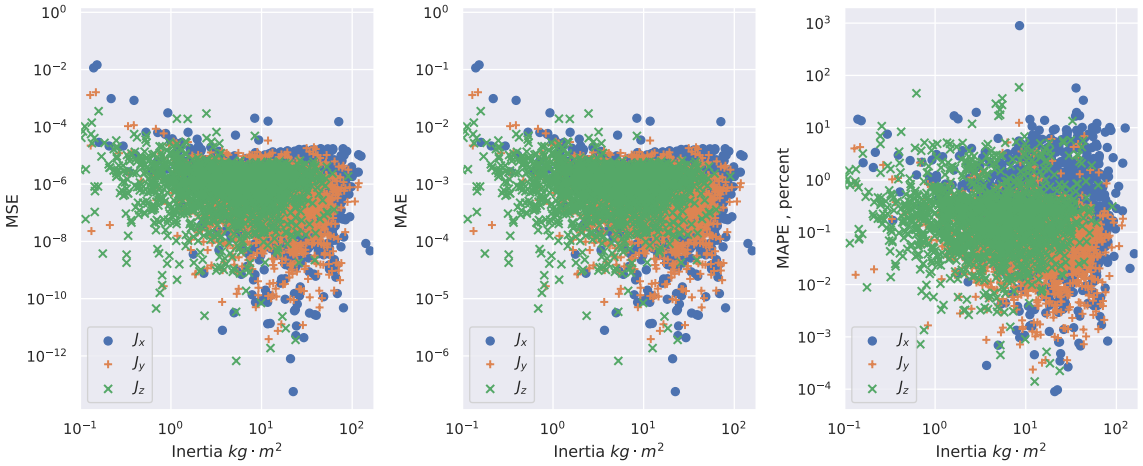

Figure 4.9. Subproblem A2, scenario 2 results

While not quite as close as in Scenario 1, Figure 4.10 shows that the distributions for Scenario 2 are very close, and difficult to distinguish.

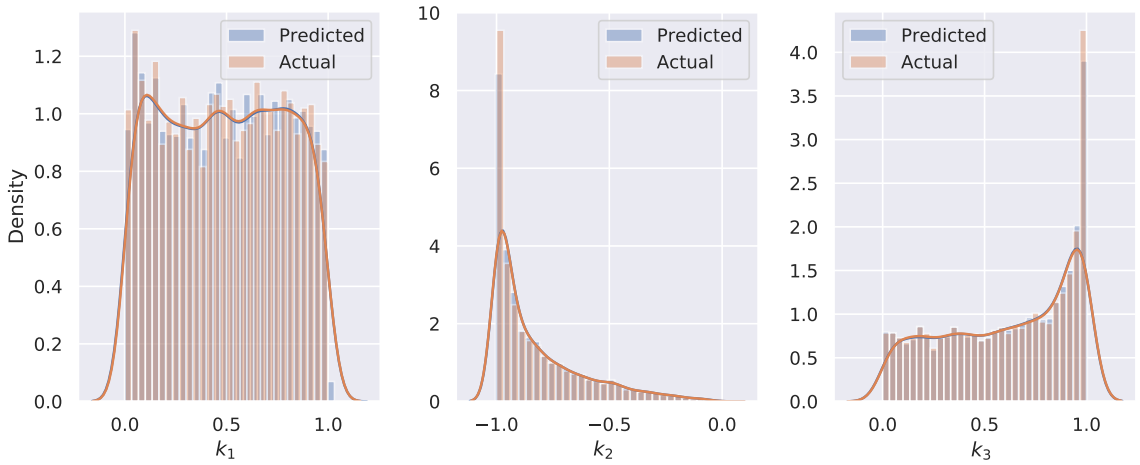

Figure 4.10. Subproblem A2, scenario 2 prediction distribution

The errors for Scenario 3 shown in Figure 4.11 are very similar to the errors in Scenario 2. Overall, the errors are small, but are larger where the MOI values are smaller.

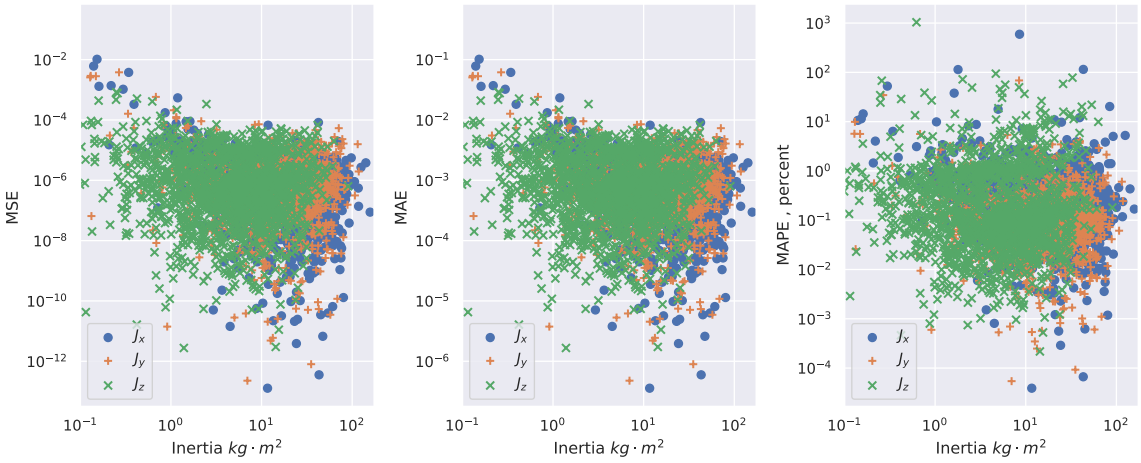

Figure 4.11. Subproblem A2, scenario 3 results

Finally, the histograms in Figure 4.12 for Scenario 3 are very similar to the histograms for Scenario 2 in Figure 4.10. The distributions match very closely, but are not quite as close as for Scenario 1.

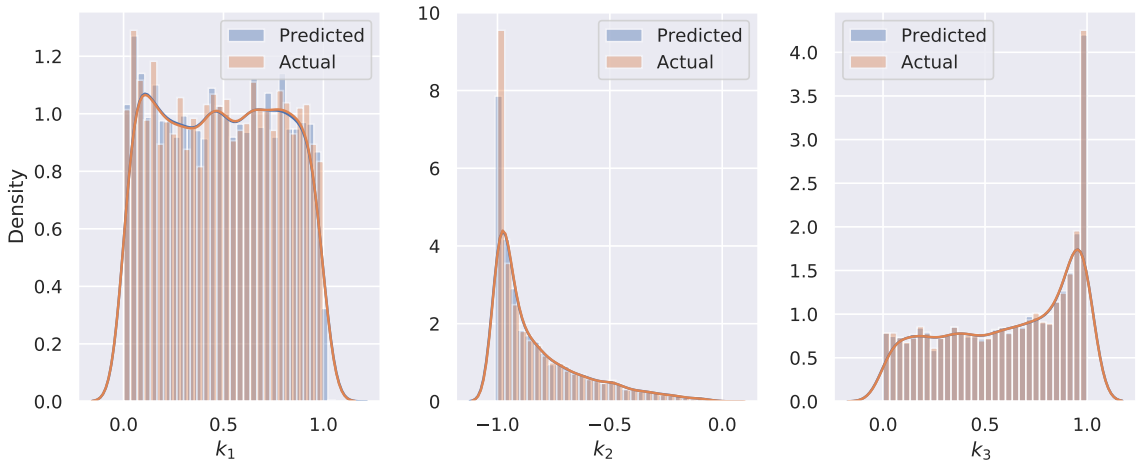

Figure 4.12. Subproblem A2, scenario 3 prediction distribution

Again, these larger errors for Scenarios 2 and 3 could be attributed to the extra features (e.g., torques) that the neural network must take into account for Scenarios 2 and 3. These errors may be reduced if the samples are longer, or if they have shorter timesteps. However, the errors for each of the three scenarios are very small, indicating that the neural network was able to predict the Smelt parameters whether or not there is a torque present, and whether or not the network has knowledge of that torque.

4.1.3 Problem A Discussion.

Scenario 1 of subproblem A1 showed that the neural network is unable to predict the MOI in torque-free motion. This is expected because the angular velocity depends only on the relative MOI ratios, and not on the absolute MOI. Scenario 1 serves as an upper bound on the errors from the neural network. Scenario 2 showed that, for this scenario, when the neural network has at least partial knowledge of the torque, it is able to estimate the MOI. This outcome is also expected because if the angular velocities and torque are known, the only unknowns are the MOI. Scenario 2 serves as a lower bound on the errors of the neural network. Scenario 3 is interesting because

although there is a torque present, the neural network has no knowledge of this torque and must predict the MOI based solely on the angular velocity. The results clearly show that the neural network performs better than in torque-free motion. The errors for Scenario 3 are much closer to the errors of Scenario 2, which suggests that, with further refinement, the neural network may be able to accurately estimate the MOI of a spacecraft when there is an unknown external torque applied.

Subproblem A2 showed that for all three Scenarios, the neural network is able to accurately estimate the Smelt parameters. This result suggests that the presence of and knowledge of external torques is not required to estimate the Smelt parameters. Interestingly, it was Scenario 1, with torque-free motion, where the neural network performed the best. One possible explanation is that in torque-free motion, the Smelt parameters are what determine the change in angular velocity, so the process of identifying the Smelt parameters was more straightforward. When there is torque present, the absolute MOI and external torques determine the change in angular velocity, so there is more information and mathematical relationships the neural network must work out.

4.2 Problem B

Problem B consists of three parts. In the first part, B1, the agent begins from an arbitrary orientation and attempts to achieve an orientation of $q_d = [0,\ 0,\ 0,\ 1]^T$. For part B2, the agent both begins and ends at arbitrary orientations. A different agent is trained for each part, and the reward history for each agent over the course of its training history is shown in their corresponding sections. The results of a randomized scenario for each agent are also shown. The third part, B3, involves using the same agents as in B1 and B2, so no new training takes place. Each agent is then subjected to a disturbance halfway through a randomized scenario.

4.2.1 Subproblem B1: Start from Arbitrary Attitude.

For subproblem B1, the spacecraft begins at a randomized orientation with randomized MOI. The agent must then control the spacecraft to achieve the orientation of $q_d = [0,\ 0,\ 0,\ 1]^T$. The agent trained for 5 million steps in the environment, resulting in the reward history shown in Figure 4.13.

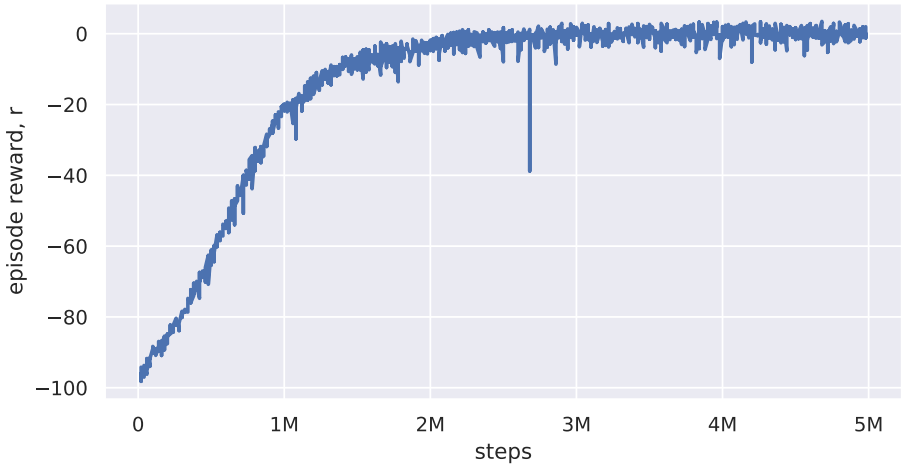

Figure 4.13. Subproblem B1 agent reward history

The large dip in the reward history between 2 and 3 million steps could be a result of the agent choosing a bad policy, but the agent quickly recovered, and the dip did not affect the overall reward history of the agent. It is also important to note that this dip happened during training while the neural network choosing the action is being tuned. When testing the agent on scenarios, no learning or adjusting of the neural network is taking place. Agent B1's reward history shows that it learned to maximize the reward it was given, but the reward history alone does not indicate whether the policy it learned allowed the agent to reach the desired orientation. To help determine the usefulness of the learned policy, a random scenario was generated, with values shown in Equation (4.1).

$$q_0 = [0.5897,\ 0.3791,\ -0.5618,\ 0.4392]^T \tag{4.1a}$$

$$q_d = [0,\ 0,\ 0,\ 1]^T \tag{4.1b}$$

$$J = \begin{bmatrix} 14.24 & 0 & 0 \\ 0 & 11.84 & 0 \\ 0 & 0 & 3.76 \end{bmatrix} \tag{4.1c}$$

This scenario was simulated for 120 seconds, and the state and control trajectories of Agent B1 are shown in Figure 4.14.

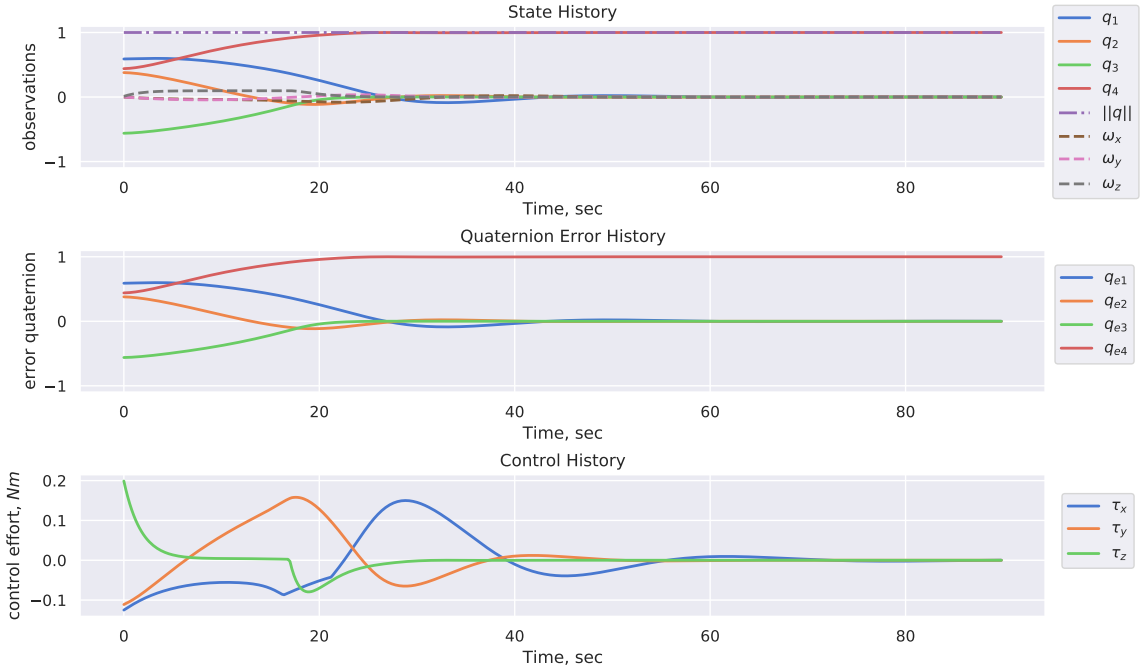

Figure 4.14. Subproblem B1 agent performance on random scenario

The agent is able to successfully control the states to the desired values, while maintaining a relatively smooth control effort. Overall, the agent has no trouble reaching the desired orientation despite having no knowledge of the MOI or of the underlying dynamics of the spacecraft. Multiple randomized simulations were run with Agent B1, and all showed very similar results, indicating that the agent learned

a useful policy for achieving the quaternion $q_d = [0,\ 0,\ 0,\ 1]^T$ starting from an arbitrary orientation.

4.2.2 Subproblem B2: Achieve Arbitrary Attitude.

The reward history for Agent B2, shown in Figure 4.15, looks similar to that of Agent B1, the main difference being that Agent B2 trained for 20 million steps rather than 5 million. Another difference is that the initial reward values are lower than the initial values for Agent B1. The initial gains in episode reward are large in the beginning of the training, going from -175 to -25 in 2.5 million steps, but it takes the remainder of the training to approach an episode reward of 0.

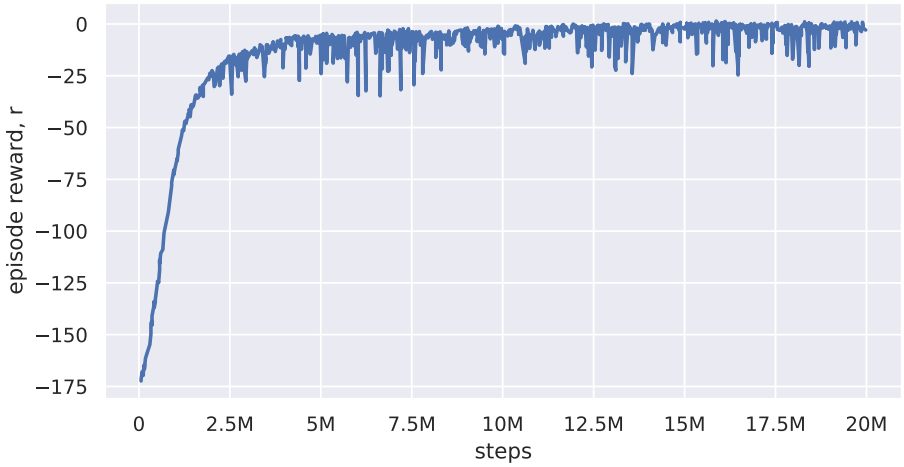

Figure 4.15. Subproblem B2 agent reward history

Just as in subproblem B1, the reward history seems to indicate that the agent converged on a policy attempting to maximize the reward function. The reward history of Agent B2 does appear to vary more than that of Agent B1 as the average episode reward begins to plateau. This is expected because the search space of the agent is larger, with the agent both starting and ending in arbitrary orientations. To

gauge whether the reward history corresponds to a useful policy, a random scenario was simulated with the values shown in Equation (4.2).

$$q_0 = [-0.7503,\ 0.0502,\ 0.0100,\ 0.6591]^T \tag{4.2a}$$

$$q_d = [0.5395,\ -0.6981,\ -0.4194,\ 0.2138]^T \tag{4.2b}$$

$$J = \begin{bmatrix} 30.05 & 0 & 0 \\ 0 & 23.19 & 0 \\ 0 & 0 & 9.18 \end{bmatrix} \tag{4.2c}$$

The scenario was propagated for 600 seconds, and the state and control histories of Agent B2 are shown in Figure 4.16.

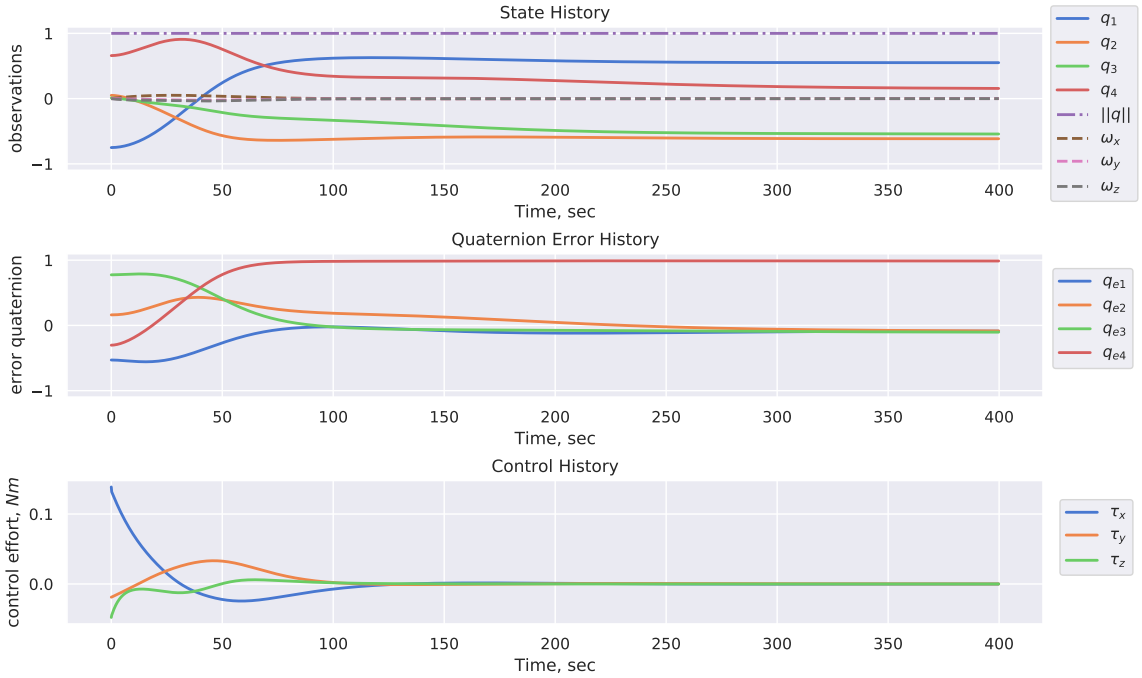

Figure 4.16. Subproblem B2 agent performance on random scenario

Agent B2 successfully achieves the desired orientation, made apparent by the error quaternion reaching $q_e = [0,\ 0,\ 0,\ 1]^T$. However, the response of the agent is much slower compared to Agent B1. It seems that Agent B2 makes relatively

large adjustments towards the beginning, and then slowly converges to the solution over time. It also appears that the values of the vector part of the error quaternion converge to a value slightly below zero instead of exactly zero. While Agent B2 achieves close to the desired orientation, it takes longer and is not as accurate as Agent B1. To be fair, Agent B2 has a much more difficult problem to solve, so the difference in performance is to be expected.

4.2.3 Subproblem B3: Simulated Collision.

In subproblem B3-1, the first agent, Agent B1, receives a new randomized MOI and an angular velocity randomly sampled from a normal distribution parameterized by $\mu = 0$, and $\sigma = 10\frac{deg}{s}$. A random scenario was generated with the values shown in Equation (4.3), where there is an abrupt change at $t = 100$ seconds.

$$q_0 = [0.2123,\ -0.2111,\ 0.9205,\ 0.2512]^T \tag{4.3a}$$

$$q_d = [0,\ 0,\ 0,\ 1]^T \tag{4.3b}$$

$$J_0 = \begin{bmatrix} 16.61 & 0 & 0 \\ 0 & 12.29 & 0 \\ 0 & 0 & 7.88 \end{bmatrix} \rightarrow\ J_{100} = \begin{bmatrix} 26.84 & 0 & 0 \\ 0 & 18.46 & 0 \\ 0 & 0 & 11.53 \end{bmatrix} kg \cdot m^2 \tag{4.3c}$$

$$\vec{\omega}_{100}^{-} = \begin{bmatrix} 0.00016 \\ 4.24 \times 10^{-6} \\ 2.32 \times 10^{-5} \end{bmatrix} \rightarrow\ \vec{\omega}_{100}^{+} = \begin{bmatrix} -0.25 \\ 0.35 \\ 0.088 \end{bmatrix} \frac{rad}{s} \tag{4.3d}$$

Figure 4.17 shows the results of this scenario. There is an abrupt change in both MOI and angular velocity at $t = 100$ seconds, but there is no abrupt change in orientation, or in other words, the quaternion is continuous.

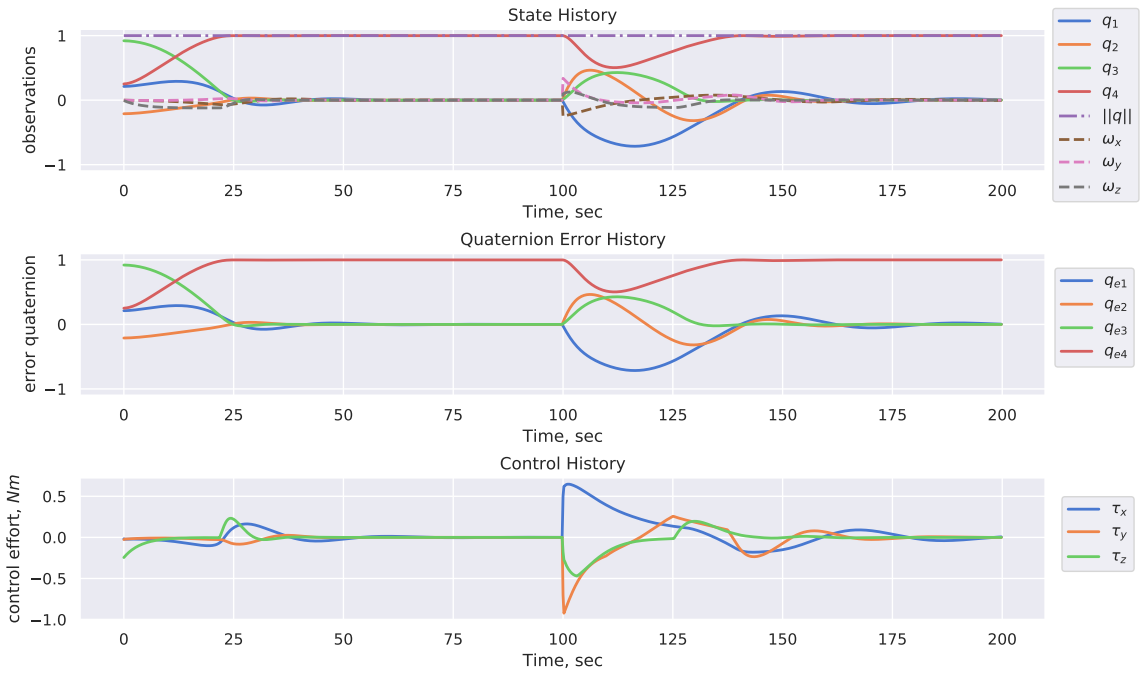

Figure 4.17. Subproblem B3-1 discontinuous change in MOI and angular velocity

Despite the abrupt changes in MOI and angular velocity, Agent B1 successfully controls the spacecraft to the desired orientation. This result is particularly interesting because the agent has no knowledge of the MOI or dynamics at any point in the episode, and at no point during training was there an abrupt change in MOI or angular velocity. During training, the MOI and angular velocity were randomized at the start of each episode, but never *during* the episode. Despite never training with abrupt changes to MOI and angular velocity, the agent is able to achieve the desired orientation both before and after the abrupt change. The ability of the trained agent to maintain control in situations for which it was never trained indicates that it learned a generalized policy that applies to more than the scenarios for which it was trained. This outcome is one of the characteristics of autonomy, and highlights the potential usefulness of model-free reinforcement learning.

A similarly randomized scenario was generated for Agent B2, but in addition to the change in MOI and angular velocity, there is also a change in desired quaternion.

While a collision event would be unlikely to change the desired quaternion, the desired quaternion was randomized to investigate the robustness of the agent because not only did Agent B2 never train with abrupt changes in MOI and angular velocity, but it also never experienced abrupt changes in desired quaternion. The initial and changed values for the randomized scenario are shown in Equation (4.4).

$$q_0 = [0.1893,\ -0.3489,\ 0.7575,\ 0.5183]^T \tag{4.4a}$$

$$q_{d,0} = [0.5790,\ -0.0728,\ 0.4157,\ 0.6976]^T \tag{4.4b}$$

$$q_{d,900} = [-0.2653,\ -0.4877,\ -0.7121,\ 0.4298]^T \tag{4.4c}$$

$$J_0 = \begin{bmatrix} 76.23 & 0 & 0 \\ 0 & 55.21 & 0 \\ 0 & 0 & 39.82 \end{bmatrix} \rightarrow J_{900} = \begin{bmatrix} 31.95 & 0 & 0 \\ 0 & 21.01 & 0 \\ 0 & 0 & 11.95 \end{bmatrix} kg \cdot m^2 \tag{4.4d}$$

$$\vec{\omega}_{900}^{-} = \begin{bmatrix} 0.0015 \\ -0.00034 \\ 0.00043 \end{bmatrix} \rightarrow \vec{\omega}_{900}^{+} = \begin{bmatrix} -0.053 \\ 0.029 \\ 0.022 \end{bmatrix} \frac{rad}{s} \tag{4.4e}$$

This scenario is simulated for 1800 seconds, and the abrupt change occurs at $t = 900$ seconds. Just as for subproblem B3-1, the continuity of the attitude quaternion is maintained. However, the desired quaternion is discontinuous because it abruptly changes halfway through the episode, but the attitude quaternion is continuous throughout the episode. The state and control histories of the randomized scenario are shown in Figure 4.18.

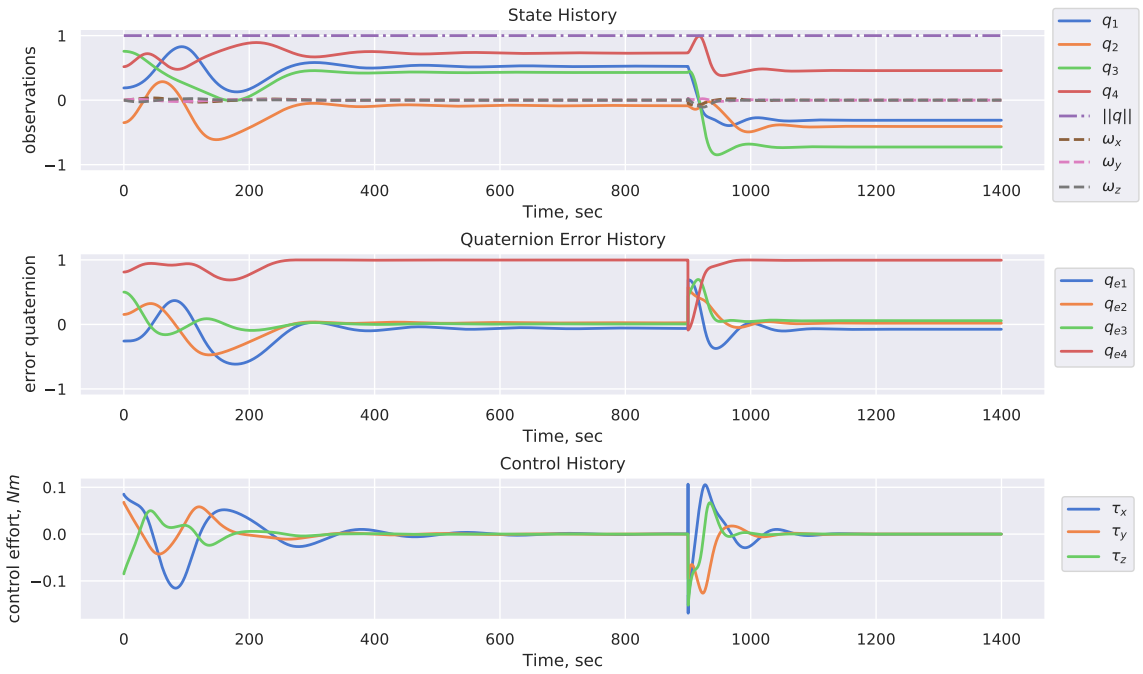

Figure 4.18. Subproblem B3-2 discontinuous change in MOI, angular velocity, and desired attitude

Looking at the first half of the episode, the results for subproblem B3-2 are similar to the results of subproblem B2. The agent is able to achieve close to the desired orientation, but the agent does not reach the desired orientation exactly. At $t = 900$ seconds, the MOI, angular velocity, and desired quaternion change. Similar to subproblem B3-1, the agent is able to control itself to reach the desired orientation despite the abrupt changes, and the second half of the episode is very similar to the first half in terms of error quaternion and control effort.

4.2.4 Problem B Discussion.

Subproblem B1 showed that a reinforcement learning agent can be trained to control a simulated spacecraft attitude control system and achieve a particular orientation. Subproblem B2 showed that an RL agent can learn to achieve an arbitrary orientation, but it may require more tuning than in the simpler case of achieving a

single orientation. Subproblem B3 showed that despite not being trained to handle abrupt changes in MOI and angular velocity, the trained agents were still able to reach their desired orientations. Not only did the agents perform well in scenarios not specifically trained on, as shown in subproblems B1 and B2, but they also performed well on an arguably different type of scenario, as shown in subproblems B3-1 and B3-2.

The main discrepancy between Agent B1 and Agent B2 is the difference in error quaternion accuracy. The inaccuracy of the error quaternion for Agent B2 indicates that the quaternion error is not penalized enough. The problem is that the error quaternion penalty is at odds with the control effort penalty because an increase in emphasis on one necessarily means a decrease in emphasis for the other. However, it is possible that there is some combination of penalties that guides the agent to a small control effort and an accurate error quaternion. There could also be a different reward function that could resolve these errors. It could also be that the hyperparameters of the policy network need to be adjusted, such as decreasing the learning rate or discount factor. Finally, it may help to change the length of each episode, currently set to 1,500 seconds, or the length of each time step, currently set to 0.2 seconds.

V. Conclusions and Future Work

5.1 Review

In summary, the goal of this thesis was to investigate the use of machine learning to help in scenarios where the model of the spacecraft may be unknown. This main idea was divided into two problems. Problem A investigated the use of recurrent neural networks to estimate the principal moments of inertia of a spacecraft given angular velocity data and in one scenario, magnetic torque data. Problem A also involved estimating the relative MOI ratios, or Smelt parameters given the same information. The neural networks for Problem A were, as expected, unable to estimate the MOI when there are no external torques acting on the spacecraft. The neural networks had much better success when the spacecraft was both experiencing an external torque and the neural network received the torque data as an input. Finally, when the spacecraft was experiencing an external torque, but the neural network did not receive the torque data as an input, the neural network did not perform as well as when the torque was known.

Whereas Problem A involved estimation, Problem B involved directly controlling the attitude of the spacecraft. In the first subproblem of Problem A, the reinforcement learning agent attempts to reach the identity quaternion starting from an arbitrary orientation. The second subproblem is similar to the first, but instead of attempting to reach the identity quaternion, the agent could be given any desired orientation. Finally, the third subproblem involved subjecting the first two agents to discontinuous changes in MOI and angular velocity, and in the case of the second agent, the desired orientation was also changed. For all scenarios, the agents were able to control the spacecraft to the desired orientation. However, the agent attempting to reach an arbitrary orientation slightly missed the desired orientation.

5.2 Insights

The results collected in this thesis demonstrate the power of machine learning techniques when applied to problems in astrodynamics. Using only data and no model information, a recurrent neural network was able to estimate the MOI and Smelt parameters of a spacecraft. This finding suggests that neural networks may be a viable option for estimating spacecraft model parameters on orbit.

The benefits offered by reinforcement learning algorithms are enticing for a number of reasons. If a reinforcement learning agent can operate without requiring knowledge of the system it is controlling or of the dynamics governing its motion, then the resulting control solution would exhibit some degree of autonomy. The agent would have the ability to react to situations it never experienced in training. As space systems become more complex, there will be a need for more autonomy, and reinforcement learning may be able to solve some of the hurdles associated with autonomous control.

Another potential benefit of training a reinforcement learning agent is that the agent can be trained in simulation before ever launching to orbit. On the ground, the agent could train on a wide variety of simulations encompassing different possible scenarios, configurations, and parameters. If trained correctly, this single agent may be generalized enough to control any number of different spacecraft configurations and fulfill a wide variety of mission requirements. It is also possible to allow the reinforcement learning agent to continue learning while on orbit in order to improve or refine the algorithm further. Another possibility is that the agent can continue to train on the ground in simulation after launch and periodically send updates to the agent on orbit.

5.3 Future Research

Generalized Body Frame. For both Problems A and B, a next logical step would be to use a generalized body frame, with both the moments and products of inertia, instead of only the principal frame. Using a generalized body frame would give a better indication as to how generalized the neural networks and reinforcement learning agents are. One possible way to create a generalized random inertia matrix would be to generate a random principal inertia matrix, and rotate it via a randomly generated quaternion. It would also be useful to study the sensitivity of the algorithms to changes in the distribution of the randomized moments of inertia. For example, would increasing the range of inertia values that could be generated decrease performance?

Noise and Gyro Bias. The data generated for Problems A and B did not include any noise. Noise could be added to the angular velocity and torque data for Problem A. For Problem B, the same perfect information could be used to update the environment, while the state information given to the reinforcement learning agent could include noise and gyro bias. Before it can be said that these machine learning techniques can be applied to real-world scenarios, the simulations the techniques are used in must more closely resemble reality. Adding noise and gyro bias would be the next logical step in increasing the fidelity of the simulations, which would then more clearly show the usefulness of these machine learning algorithms.

Compare Results with Baselines. Another important area for future work would be to directly compare the machine learning techniques to different baselines. For example, the recurrent neural network estimation of MOI could be compared to a least-squares or Kalman filter solution using the same scenario. The deep reinforcement learning control algorithm could be compared to an LQR, PID, or MPC solution for a scenario that includes an abrupt change in MOI and angular velocity. Without

direct comparisons it is difficult to assess the usefulness of these algorithms in the real world.

Automate Hyperparameter Tuning. Because both Problems A and B required a significant amount of tuning, it is fair to say that the final hyperparameters that were chosen are not at their ideal values. One possible remedy for this issue is to make use of a software package that automates hyperparameter tuning. The chosen software package would ideally be compatible with the software developed in this thesis, but some adjustments will inevitably be required. By automating the process of tuning hyperparameters, a better performing neural network or RL agent can much more easily be found.

Real-Time Processing and Close the Loop. For Problem A specifically, instead of processing data in batches of 30 seconds, it may be more beneficial to process the data in real time. Not only would this be more realistic, but the change in estimated MOI values over time may offer some insight into the neural network's estimation process. It would also be insightful to include the neural network estimator in a closed-loop control system to determine whether its predictions enable the controller to function properly and whether the resulting system would be stable.

Improve Accuracy. While the results for Problem B are very promising, orientation accuracy for Agent B2 leaves much to be desired. Zeroing out the vector part of the error quaternion would be the next step towards improving the performance of Agent B2. The hyperparameter tuning software mentioned earlier may improve the performance, but if not, the most likely cause of the performance issues is the reward function. Changing the proportion of control penalty to error quaternion penalty may simply need to be adjusted. However, fixing this issue may require reformatting the way the penalties are calculated in the first place. For example, instead of penalizing the sum of the control at each time step, it may be beneficial to penalize some

combination of the derivative and integral of control. Also, instead of rewarding the agent for moving towards the goal and penalizing for moving away, it may be useful to instead penalize the magnitude of the error quaternion.

Add Keep-Out Zones. In some cases, there are certain attitudes that should be avoided. For example, it may be prudent to keep a sensitive camera out of direct sunlight. Incorporating keep-out zones would be another major step forward for the reinforcement learning agent. The most likely solution would be to include steep penalties for entering a keep-out zone. However, there are a few elements that require some forethought. First, the agent must be aware of the current keep-out zone, so finding a way to parameterize the keep-out zone that is easy for the agent to understand is essential. Next, if arbitrary keep-out zones are desired, there must be a way to randomly generate them so the agent can see a wide variety during training. However, the randomly generated initial and desired orientations must not fall within the keep-out zones. It may make sense to allow the initial orientation to fall within the keep-out zone so the agent can learn the orientation is undesirable, but the desired orientation cannot be in the keep-out zone or the agent may not learn how to deal with the conflicting priorities. Adding more information for the agent to learn will mean the agent will take longer to train and may have a more difficult time converging to a useful policy.

Expand to 6-DOF Environment. One final thought for future work involves expanding the environment from 3-DOF, rotation only, to a 6-DOF environment including both rotation and translation. Problem B demonstrated the potential for reinforcement learning to be useful for attitude control, so it may be useful to determine whether reinforcement learning could also be useful for controlling orbital motion. Having one agent control both the rotational and translational motion may prove to be difficult, so one possible alternative would be train one agent to control

the attitude and a second agent to control the orbit. If using coupled rotational and translational dynamics, it would be interesting to study the difference between training the agents together vs training the agents separately. Perhaps if trained independently, the agents would compete for control, whereas if trained together, the agents may learn to efficiently work together.

5.4 Conclusion

This thesis developed two proof-of-concept machine learning solutions: one to estimate spacecraft moments of inertia and another to control the attitude of a spacecraft. A recurrent neural network was used for the first problem and was able to estimate the moments of inertia for some scenarios, and the neural network was also able to estimate the relative moment of inertia ratios for all the scenarios tested. A deep reinforcement learning algorithm successfully controlled the attitude of a spacecraft to reach the desired orientation. When tested on a few of the scenarios, the reinforcement learning agent showed promising results, in some ways exhibiting behaviors associated with autonomy. The reinforcement learning agent was able to generalize its solution to situations in which it was never trained. The agent experienced abrupt changes, and despite having no knowledge of the spacecraft parameters or attitude dynamics, the agent still reached the desired orientation. These behaviors suggest that some level of autonomy has been achieved on the part of the reinforcement learning agent. Although this research shows promising results, there is still much work that can be done to both improve the algorithms and demonstrate usability on real-world systems. The novel approaches to MOI estimation and attitude control presented in this research give confidence to the notion of using artificial intelligence to tackle complex problems in astrodynamics.

Bibliography

[1] J. Burl, *Linear Optimal Control*. Addison-Wesley, 1999.

[2] F. Chollet, *Deep Learning with Python*. Manning Publications Company, 2017.

[3] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning, The MIT Press, second ed., 2018.

[4] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[5] United States Space Force, "Spacepower: Doctrine for Space Forces," *Space Capstone Publication*, 2020. www.spaceforce.mil.

[6] White House, "National Security Strategy of The United States of America," 2017.

[7] White House, "National Defense Strategy of The United States of America," 2018.

[8] U.S. Air Force, "U.S. Air Force Science and Technology Strategy for 2030 and Beyond," 2019.

[9] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education Inc., 2010.

[10] S. S. Tavallaey and C. Ganz, "Automation to autonomy," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 31–34, 2019.

[11] G. Snodgrass, "AI Defeats Air Force Pilot in Head-To-Head Competition," *Forbes*, 2020.

[12] T. Senator, "Physics of Artificial Intelligence (PAI)." `https://www.darpa.mil/program/physics-of-artificial-intelligence`.

[13] DARPA, "Techniques for Machine Vision Disruption (TMVD)." `https://beta.sam.gov/opp/3d75659b52154edd8e74a6e86cc519d6/view`, 2020.

[14] K. Kearns, "DoD Autonomy Roadmap," 2018. Download: `https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2018/science/Kearns.pdf`.

[15] D. A. Vallado and W. D. McClain, *Fundamentals of Astrodynamics and Applications*. Microcosm Press, 4 ed., 2013.

[16] D. Kunz, *Intermediate Dynamics for Aeronautics and Astronautics: Second Edition*. Independently Published, 2019.

[17] F. L. Markley and J. L. Crassidis, *Fundamentals of Spacecraft Attitude Determination and Control*. Space Technology Library, pringer, 2014.

[18] P. Hughes, *Spacecraft Attitude Dynamics*. Dover Books on Aeronautical Engineering, Dover Publications, 2012.

[19] B. Wie, *Space Vehicle Dynamics and Control*. AIAA education series, American Institute of Aeronautics and Astronautics, 2008.

[20] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*. AIAA Education, American Institute of Aeronautics and Astronautics, Inc., second ed., 2009.

[21] B. Friedland, *Control System Design: An Introduction to State-Space Methods*. Dover Publications, Inc., 2005.

[22] K. Ogata, *Modern Control Engineering*. Prentice Hall, 5 ed., 2010.

[23] D. E. Kirk, *Optimal Control Theory: An Introduction*. Dover Publications, Inc., 1998.

[24] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics, Springer New York, 2013.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] G. Cybenko, "Degree of Approximation by Superpositions of a Sigmoidal Function," *Approx. Theory its Appl.*, vol. 9, no. 3, pp. 17–28, 1989.

[27] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[29] OpenAI, "OpenAI Five." `https://blog.openai.com/openai-five/`, 2018.

[30] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving Rubik's Cube with a Robot Hand," *arXiv preprint*, 2019.

[31] Zai and Brown, *Deep Reinforcement Learning In Action*. Manning Publications Company, 2020.

[32] B. D. Little and C. E. Frueh, "Space Situational Awareness Sensor Tasking: Comparison of Machine Learning with Classical Optimization Methods," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 2, pp. 262–273, 2020.

[33] E. A. Youmans and F. H. Lutze, "Neural Network Control of Space Vehicle Intercept and Rendezvous Maneuvers," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 1, pp. 116–121, 1998.

[34] N. Coulter and H. Moncayo, "An Online Machine Learning Paradigm for Spacecraft Fault Detection," in *AIAA Scitech 2021 Forum*, 2021.

[35] G. Chen and J. B. Cruz, "Genetic Algorithm for Task Allocation in UAV Cooperative Control," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003.

[36] J. A. Hess, *Adaptive Estimation and Heuristic Optimization of Nonlinear Spacecraft Attitude Dynamics*. PhD thesis, Air Force Institute of Technology, 2016.

[37] J. A. Keim, A. Behcet Acikmese, and J. F. Shields, "Spacecraft Inertia Estimation via Constrained Least Squares," in *2006 IEEE Aerospace Conference*, 2006.

[38] J. Thienel, R. Luquette, and R. Sanner, "Estimation of Spacecraft Inertia Parameters," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008.

[39] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[40] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement Learning for UAV Attitude Control," 2018.

[41] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy optimization," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 523–533, 2019.

[42] X. Wang, G. Wang, Y. Chen, and Y. Xie, "Autonomous Rendezvous Guidance via Deep Reinforcement Learning," in *Chinese Control and Decision Conference*, 2020.

[43] K. Hovell and S. Ulrich, "On Deep Reinforcement Learning for Spacecraft Guidance," in *AIAA Scitech 2020 Forum*, 2020.

[44] MATLAB, *version 9.7.0 (R2020a)*. Natick, Massachusetts: The MathWorks Inc., 2020.

[45] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[46] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[47] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. Software available from `tensorflow.org`.

[48] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.

[49] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[50] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable Baselines." `https://github.com/hill-a/stable-baselines`, 2018.

[51] M. A. Peck, "Uncertainty Models for Physically Realizable Inertia Dyadics," *The Journal of the Astronautical Sciences*, vol. 54, pp. 1–16, 03 2006.

[52] S. Miller and D. Childers, *Probability and Random Processes*. Elsevier Inc., second ed., 2012.

[53] F. Rimrott, *Introductory Attitude Dynamics*. Springer-Verlag, 1989.

[54] K. Shoemake, "Uniform Random Rotations," in *Graphics Gems III* (D. Kirk, ed.), ch. 3.6, Academic Press, Inc., 1992.

[55] N. Enders, J. Curro, J. Hess, and R. Cobb, "Spacecraft Moment of Inertia Estimation via Recurrent Neural Networks," in *AIAA/AAS Astrodynamics Specialist Conference*, 2020.

[56] J. Dormand and P. Prince, "A family of embedded Runge-Kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19 – 26, 1980.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	Aug 2019 – Mar 2021

4. TITLE AND SUBTITLE

Deep Reinforcement Learning Applied to Spacecraft Attitude Control and Moment of Inertia Estimation via Recurrent Neural Networks

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Enders, Nathaniel, A, 2d Lt, USSF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENY-MS-21-M-298

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved For Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This study investigated two distinct problems related to unknown spacecraft inertia. The first problem explored the use of a recurrent neural network to estimate spacecraft moments of inertia using angular velocity measurements. Initial results showed that, for the configuration examined, the neural network can estimate the moments of inertia when there is a known external torque. The second problem trained a reinforcement learning agent, via proximal policy optimization, to control the attitude of a spacecraft. The results demonstrated that reinforcement learning may be a viable option for guidance and control solutions where the spacecraft model may be unknown. The trained agents displayed a degree of autonomy with their ability to recover from events never experienced in training.

15. SUBJECT TERMS

Spacecraft Attitude Control; Artificial Intelligence; Machine Learning; Deep Reinforcement Learning; Moments of Inertia; Deep Learning; Autonomy

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Maj Joshuah A. Hess, AFIT/ENY
U	U	U	UU	106	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4713 joshuah.hess@afit.edu