

Design and Evaluation of an AI-Driven Autonomus control System for Deep-Space Simulation

Supervised by Ogbonnaya Anicho

By Tinomudaishe Nigel Chinembiri – 22004221

May 2025

**Department of Mathematics and Computer Science**

# Abstract

This research presents the design and implementation of an AI-driven autonomous control system, integrated into a custom-built space simulation environment. The motivation stems from the critical limitations of traditional mission control systems which becomes increasingly problematic in deep-space missions. The system addresses these issues through a modular architecture composed of three independently developed components, a CUDA-accelerated physics simulation engine, a transformer-based reinforcement learning AI agent, and a spacecraft firmware interface for real-time monitoring and control.

The AI agent uses a transformer-based Q-learning model to identify mission-appropriate behaviours based on sensor data sequences from the past. It uses a hierarchical memory architecture that includes Short-Term Memory (STM), Intermediate Memory (IM), and Long-Term Memory (LTM), allowing the agent to retain contextual knowledge over different time horizons for better informed decision-making. A bidirectional ZeroMQ-based Interposes Communication (IPC) architecture facilitates communication between subsystems, allowing the AI to receive telemetry and perform actions in sync with the simulation.

The space environment replicates realistic orbital mechanics by using Newtonian gravity and Barnes-Hut approximations, as well as spaceship thrust modelling, Runge-Kutta-Fehlberg integration, and basic collision detection. The system is examined modularly before being merged, with physical precision, AI response, and subsystem compatibility being the primary evaluation criteria.

The results show strong performance in areas including GPU-accelerated physics, memory reliability, and mission-level decision convergence. However, full-system testing found limitations in decision alignment across components, indicating areas for improvement in state consistency and feedback integration.  
  
The project demonstrates the viability of a scalable, AI-powered autonomous control system for future space missions and lays the groundwork for future research in cognitive spaceflight systems.

Table of Contents

[Abstract 2](#_Toc198266270)

[1. Introduction: 6](#_Toc198266271)

[1.1 Background 6](#_Toc198266272)

[1.1 Research Questions and Expected Contributions: 7](#_Toc198266273)

[1.1.1 How can an AI-based system enhance spacecraft autonomy in deep-space missions? 7](#_Toc198266274)

[1.1.2 What are the potential limitations and risks of an AI-controlled spacecraft system, and how can they be mitigated 7](#_Toc198266275)

[1.2 Aims and Objectives: 8](#_Toc198266276)

[2. Literature Review: 9](#_Toc198266277)

[2.1 Foundation theory of RL: 9](#_Toc198266278)

[2.2 Transformers in RL 10](#_Toc198266279)

[2.3 Exploration & exploitation strategies 10](#_Toc198266280)

[2.4 Existing space AI systems: 11](#_Toc198266281)

[2.5 Fault detection & anomaly response 11](#_Toc198266282)

[2.6 Memory Architectures in AI 12](#_Toc198266283)

[2.7 Interposes Communication (IPC) and Systems Integration 12](#_Toc198266284)

[2.8 Simulation & Physics Engines for RL 13](#_Toc198266285)

[3. Methodology: 14](#_Toc198266286)

[3.1 Development of the custom space simulator environment: 15](#_Toc198266287)

[3.1.1 CUDA-accelerated physics engine 16](#_Toc198266288)

[3.1.2 Bidirectional Interposes Communication (IPC) 19](#_Toc198266289)

[3.2 AI-driven autonomous spacecraft control systems: 22](#_Toc198266290)

[3.2.1 Transformer-Based Reinforcement Learning Model: 23](#_Toc198266291)

[3.2.2 Hierarchical memory system 25](#_Toc198266292)

[3.2.3 Anomaly Detection System 27](#_Toc198266293)

[3.2.4 Natural Language Command Interpretation 29](#_Toc198266294)

[3.2.5 Contextual Mission Phase Awareness 31](#_Toc198266295)

[4. Implementation overview 32](#_Toc198266296)

[4.1 The AI agent layer (Python) 33](#_Toc198266297)

[4.1.1 Transformer-Based Architecture 34](#_Toc198266298)

[4.1.2 Dynamic Contextual Reasoning via Sequence Length Adaptation 35](#_Toc198266299)

[4.1.3 Inference and Action Execution pipeline 35](#_Toc198266300)

[4.1.4 Training Regime 37](#_Toc198266301)

[4.2 Interposes Middleware (ZeroMQ) 38](#_Toc198266302)

[4.2.1 Layered Architecture 38](#_Toc198266303)

[4.3 Physics and Environmental Simulation Engine (C++/CUDA) 40](#_Toc198266304)

[4.3.1 Architecture and core design 41](#_Toc198266305)

[4.3.2 Object-Oriented Layering 41](#_Toc198266306)

[4.3.3 GPU-Accelerated Physical Behaviours and Dynamics 42](#_Toc198266307)

[4.4 Monitoring and Visualisation System (OpenGL + Logging) 45](#_Toc198266308)

[4.4.1 OpenGL-Based Visualisation System 45](#_Toc198266309)

[4.4.2 Logging and Introspection Framework 45](#_Toc198266310)

[4..5 Implementation Overview – Summary 46](#_Toc198266311)

[5. Results and Discussion 47](#_Toc198266312)

[5.1 Evaluation Criteria 47](#_Toc198266313)

[5.2 Space Environment Simulation Evaluation 47](#_Toc198266314)

[5.2.1 Accuracy and Stability of Newtonian Gravity 48](#_Toc198266315)

[5.2.2 RKF45 Integration Performance 48](#_Toc198266316)

[5.2.3 GPU Performance and Computational Throughput 48](#_Toc198266317)

[5.2.4 Observed Issues in Barnes-Hut Implementation 48](#_Toc198266318)

[5.2.5 Summary of findings 49](#_Toc198266319)

[5.3 AI Agent Evaluation 49](#_Toc198266320)

[5.3.1 Mission Success and Performance Metrics 49](#_Toc198266321)

[5.3.2 Decision-Making and Behavioural Patterns 50](#_Toc198266322)

[5.3.3 Anomaly Handling and Environmental Adaptation 50](#_Toc198266323)

[5.3.4 Training Regime and Policy Convergence 50](#_Toc198266324)

[5.3.5 Hierarchical Memory System Usage 51](#_Toc198266325)

[5.3.6 Anomaly Detection and Hazard Response 52](#_Toc198266326)

[5.3.7 Summary of AI Agent Performance 53](#_Toc198266327)

[5.4 Spacecraft Firmware Evaluation 53](#_Toc198266328)

[5.4.1 Test Overview 53](#_Toc198266329)

[5.4.2 Command Reliability 54](#_Toc198266330)

[5.4.3 Fault Detection and Recovery 54](#_Toc198266331)

[5.4.4 Telemetry Buffering and Synchronisation 54](#_Toc198266332)

[5.4.5 Evaluation Summary 55](#_Toc198266333)

[5.5 Integrated System Evaluation 56](#_Toc198266334)

[5.5.1 Evaluation Metrics and Results 56](#_Toc198266335)

[5.5.2 Real-Time Responsiveness 56](#_Toc198266336)

[5.5.3 IPC Interoperability 56](#_Toc198266337)

[5.5.4 Autonomous Decision Alignment 56](#_Toc198266338)

[5.6 Summary of Findings 57](#_Toc198266339)

[6. Conclusion 58](#_Toc198266340)

[References 59](#_Toc198266341)

# 1. Introduction:

## 1.1 Background

In modern-day space exploration, the ability to operate spacecraft autonomously is an essential requirement for long-duration, deep-space navigation and efficient resource management. Traditional spacecraft operation relies on a ground control system that communicates with the spacecraft, where ground control serves as the nerve centre of the spacecraft for decision-making. This approach has been effective for many years. However, it comes at significant costs in terms of time and resource requirements.

One of the unique challenges faced by traditional spacecraft operations in deep space missions is the significant communication delays and reliance on constant communication between Earth and the spacecraft. Electromagnetic signals used in traditional radio communications travel at the speed of light, yet the signal delay increases substantially with distance.

For instance, a typical signal transmission to Mars can take up to 20 minutes to reach its destination, with a round-trip communication taking approximately 40 minutes. For more distant targets like Jupiter or Saturn, these delays extend into hours or even days (McBrayer et al., 2021). Atmospheric conditions and celestial interference can further increase these delays.

These communication constraints pose serious problems for real-time decision-making during critical mission phases that require immediate responses based on sensor data. An autonomous system capable of independent decision-making becomes essential when communication with Earth is impractical for time-sensitive operations.

Beyond communication delays, traditional mission control systems face inherent limitations. Human operators, while skilled and adaptable, are susceptible to fatigue, cognitive overload, and stress-induced errors during extended missions. In contrast, an AI-driven autonomous system can operate for extended periods, analysing vast amounts of sensor data in real-time and making consistent decisions unaffected by human limitations. However, AI traditionally struggle with flexibility in novel situations where human intuition excels.

To address these challenges, this project presents an AI system capable of executing missions autonomously with minimal human intervention. The system leverages a transformer-based deep learning model for sequence-based reasoning, reinforcement learning (RL) for decision optimisation, and anomaly detection systems to enhance adaptability. The AI is designed to function within a close-to-realistic environment simulation where it can interact directly with a physics-based spacecraft model capable of executing complex manoeuvres under simulated gravity and gathering environmental data using onboard sensors.

Furthermore, the AI incorporates a hierarchical memory system to enhance its ability to reason about past events to make better decisions and improve long-term learning capabilities required in space exploration. This memory system consists of three:

1. Short-Term Memory (STM) - Stores real-time sensor data for immediate decision-making.
2. Intermediate Memory (IM) – Acts as a buffer between STM and LTM to allow the AI to recognise patterns and trends from recent missions.
3. Long-Term Memory (LTM) – Use deep learning to retain previous mission experiences and improve future decision-making.

By integrating these components into the system, it aims to enable fully autonomous spacecraft operations, reducing the need for a constant reliance on human oversight while allowing the system to continuously learn and adapt from its experience.

## 1.1 Research Questions and Expected Contributions:

This research is driven by the keen interest to enhance autonomy in deep-space missions through AI-enabled control systems. The following two key questions guide the development and evaluation of the proposed architecture of AI-driven control systems:

### 1.1.1 How can an AI-based system enhance spacecraft autonomy in deep-space missions?

This question is meant to address the core aim of the project, which is to develop an AI system capable of operating a spacecraft with minimal human intervention. The project explores this by designing and implementing a transformer-based reinforcement learning agent that integrates mission data, system feedback, and long-term memory structures to make real-time decisions.

The expected outcomes are to demonstrate a modular, learning-enabled control system capable of executing navigation, anomaly handling, and resource management autonomously. It contributes a novel AI architecture tailored for space environments, incorporating context-awareness, memory hierarchies, and action parameterisation for fine-grained control.

1.1.2 What are the potential limitations and risks of an AI-controlled spacecraft system, and how can they be mitigated?

This question examines the safety and integration challenges associated with deploying AI in autonomous control systems and evaluates fault tolerance mechanisms, memory safeguards, and failure detection systems. Special attention was given to the integration between the AI agent, simulation environment, and spacecraft firmware.

This study will key risks in AI-driven autonomy, such as system fragility under integration delays, lack of real-time guarantees, and difficulty in coordinating distributed modules. It contributes practical mitigation strategies, including safety state escalation protocols and intermediate memory buffering for fault recovery.

## 1.2 Aims and Objectives:

The aim of the project is to design and develop an autonomous AI-driven control that is capable of performing mission-critical tasks in a deep-space simulation environment while minimising reliance on human intervention and enhancing adaptability, safety, and long-term learning capabilities.

The objective of the project is to

* Develop a modular, Transformer-based reinforcement learning agent that can reason over sequences of sensor data and autonomously select optimal spacecraft actions in varying mission contexts.
* Integrating a hierarchical memory system to support short-term responsiveness, medium-term event tracking, and long-term experiential learning within the AI control architecture.
* Construction a CUDA-accelerated physics simulation engine to model gravitational interactions, spacecraft propulsion, collision detection, and dynamic space environments.

# 2. Literature Review:

This literature review examines autonomous decision-making in high-risk environments, particularly deep space missions characterised by uncertainty, isolation, and communication latency.

The system systematically explores five interlinked domains essential: reinforcement learning (RL) and its applications in autonomous systems, such as the transformer architecture adapted for temporal and mission-critical decision-making with the integration of hierarchical memory systems for supporting strategic cognition, as well as simulation technologies for physics-rich environments and interposed communication mechanisms that will enable the ability for synchronisation between the physical environment, spacecraft firmware, and artificial intelligence (AI). Furthermore, this review will examine the approaches to anomaly detection, natural language command interpretation, and safety considerations critical to deploying AI. These areas together will form the technical foundation for the development of a basic AI framework capable of navigating and managing autonomous spacecraft within a custom environment.

## 2.1 Foundation theory of RL:

Reinforcement Learning (RL) is a key paradigm within machine learning in which an AI agent learns to make optimal decisions through interacting with an environment by having the agent receive observations and feedback in the form of rewards. Though RL is an iterative process, where the agent will refine its policy by mapping it from the perceived states to actions with the goal of maximising future returns (Sutton and Barto, 2014). The core of RL lies in the Markov Decision Process (MDP), a mathematical framework introduced by Bellman in 1957, which provides a model for sequential decision-making under the assumption of full observability and a stationary environment (Even-Dar, Kakade and Mansour, 2004).

However, the fact is that it is rare for the real-world environment to be fully observable and stationary, especially in a high-risk environment such as space, leading to the adoption of Partially Observable Markov Decision Processes (POMDPs), which is an extension of MDPs to handle uncertainty in the state information. In POMDPs the agent receives observations that only partially reflect the state of the environment (Spaan, n.d.) (Doshi-Velez, 2009).

## 2.2 Transformers in RL

The integration of transformers into RL has garnered significant attention in recent years as researchers seek to overcome the various challenges inherent to traditional RL approaches, such as difficulties in handling long-term dependencies and partial observability. Transformers were originally designed for natural language processing (Vaswani et al., 2017); however, transformers have since demonstrated their effectiveness in multiple domains, including computer vision, robotics, and sequential decision-making. In the context of RL, they can model the temporal relationships through an attention mechanism better than the recurrent networks, which often struggle with vanishing gradients and limited memory capacity.

One notable application can be seen in autonomous driving , where a Deep Reinforcement Learning Navigation via Decision Transformer (DRLNDT) framework utilises a transformer neural network to model temporal dependencies in observations and actions (Ge et al., 2024). This approach addresses the problem posed by the partial observations in an urban environment, enabling the agent to have improved decision-making capabilities in autonomous vehicles without prior knowledge of the environment. Illustrates the broader potential of transformer-based models to improve sample efficiency, policy generalisation, and adaptability in reinforcement learning applications.

Although transformers offer strong architecture for decision-making, their success mostly relies on balancing exploration against exploitation techniques, especially in uncertain surroundings like space missions.

## 2.3 Exploration & exploitation strategies

The efficiency of learning algorithms in dynamic environments is influenced by the balance between exploration and exploitation techniques, which are essential elements in the field of artificial intelligence, particularly in the field of reinforcement learning.

One area of research that highlights the exploration-exploitation dilemma was the integration of RL techniques into mobile robotics. The researchers indicate that various RL algorithms can be employed to design exploration strategies for both single- and multi-robot systems. The strategies address challenges such as the curse of dimensionality and reward shaping, which are inherent in complex environments (Garaffa et al., 2021).

Furthermore, one approach to managing exploration and exploitation is through the development of specialised architectures, such as the Controller Exploitation-Exploration (CEE) reinforcement learning architecture. Had the architecture integrate a controller with fast-tracked learning and an actor-critic model to achieve near-optimal policies. The controller would utilise Kullback–Leibler divergence to assess whether the new strategies would outperform the existing ones, thereby facilitating a balance between exploration and exploitation (Asiain, Clempner and Poznyak, 2018).

## 2.4 Existing space AI systems:

In recent years AI has been increasingly integrated into various aspects of modern-day life, and that includes specialised areas like space and aerospace engineering, with several existing systems demonstrating how AI can be implemented into these systems. AI, as a technology, is being utilised for intelligent design in space systems to help enhance mission management and enable spacecraft to adapt actively to complex missions and harsh environments. These systems also enable the ability for intelligence to collaborate among the groups of spacecraft and support autonomous operations during deep space exploration (Huan and Shijin,2023).

When it comes to the context of space missions, AIs are being employed in areas where they can address significant challenges such as mission design, Earth observation, and the management of data processing. The utilises of these AI techniques have been gaining attention, to the point that a survey of relevant problems and proposed solutions was done that aims to improve mission efficiency and effectiveness (Russo and Lax, 2022).

## 2.5 Fault detection & anomaly response

Fault detection and anomaly response are critical components in various industrial applications, especially in the context of creating an automated spacecraft control system that needs to be able to understand its overall health through monitoring and prediction of when a system is about to fail and needs to be changed to backup systems.

In these types of systems utilising autoencoders have highlighted their effectiveness in identifying and addressing operational issues. Like, for example, one significant study focuses on wind turbines, where autoencoder-based neural networks are employed to help improve the fault detection processes of the turbines. This approach leverages the autoencoder's ability to reconstruct input data, allowing for the identification of deviations from normal operational patterns. Emphasising how it can improve the operational efficiency of wind turbines with early fault detection (Maxion, 2025).

Additionally, when it comes to the context of collaborative robots, it demonstrated how the application of multiple types of autoencoders for anomaly detection can utilise operation data to produce results that indicate that the autoencoder can effectively identify anomalies with high accuracy and provide a robust solution for real-time condition monitoring (Ayankoso et al., 2024).

## 2.6 Memory Architectures in AI

Memory architecture in AI includes Long Short-Term Memory (LSTM) systems, inspired by human cognitive processes, have become a foundational element in the development of intelligent systems, particularly those that operate in dynamic or temporally extended environments. Early AI systems relied often on symbolic representations and static memory structures such as rule-based databases or finite-state machines, which limited their ability to adapt to new experiences or remember long-term patterns. So, with the emergence of neural-based memory models, it marked a sudden change. Especially with the introduction of the LSTM network (Hochreiter & Schmidhuber, 1997), it provided a mechanism for the AI to process sequential data by maintaining internal hidden states that evolve over time. However, they had a limitation in trying to capture long-range dependencies, leading to the development of attention-based models, culminating in the transformer architecture (Vaswani et al., 2017), which enabled efficient learning over longer sequences by leveraging self-attention mechanisms.

## 2.7 Interposes Communication (IPC) and Systems Integration

Interposed communication plays a significant role in integrating various systems, with artificial intelligence (AI), particularly when integrating AI with robotics. The publication on ns3-ai discusses a newly designed module that enhances IPC between the ns-3 network simulator and multiple Python-based AI frameworks. The module utilises a shared memory approach for the IPC to achieve transfer speeds up to 100 times faster than previous implementations, such as ns3-gym, thereby facilitating efficient data exchange crucial for performance evaluation in AI-driven scenarios (Yin et al., 2020).

In the context of robotics, researchers created a Broker IPC solution that presented a flexible framework that allows for multiple processes within a robot system to communicate without interfering with each other's operations. Which is particularly beneficial in complex robotic systems where processes may terminate unexpectedly, ensuring that communication remains reliable (McNaughton et al., 2005).

## 2.8 Simulation & Physics Engines for RL

N-body simulations represent a critical technique in computational astrophysics for modelling space environments, enabling the simulation of systems with many interacting celestial bodies governed by gravitational forces. These particles can represent stars, galaxies, or other celestial bodies, and their interactions are governed by Newton’s law of gravitation. The classical N-body problem entails the computing of the gravitational forces between every pair of bodies, leading to a computational complexity of O(N^2), which is prohibitive due to the increase in N. To address this problem, a variety of the approximation algorithms was developed. The most prominent algorithm is the Barnes-Hut algorithm, which reduces the complexity to O(N log N) by hierarchically grouping distant particles and approximating their collective effect (Aarseth, 2003).

While other algorithms approach similarly exploit spatial structure to improve scalability. These algorithms are rooted in mathematical modelling, leveraging tools from classical mechanics, numerical integration, and potential theory to understand the intricate behaviour of astrophysical systems. These algorithms have proven to be particularly valuable in simulating phenomena such as planetary system formation, star cluster dynamics, and galactic evolution (Aarseth, 2003). However, despite such advances, achieving efficient performance for large-scale simulations remains an ongoing challenge, especially when deploying these algorithms in parallel or distributed computing environments (Brandt, 2022).

One of the mathematical algorithms used to improve and solve the N-body problem is the Runge-Kutta-Fehlberg (RKF45) method because in an N-body simulation, it is addressed in the context of solving initial value problems for systems of ordinary differential equations (ODEs). The RKF45 algorithm is highlighted as an adaptive step-size numerical method that can effectively manage the complexities associated with N-body problems. This technique aims to eliminate catastrophic cancellations, which can occur during the numerical integration of ODE systems, thereby enhancing the stability and accuracy of the simulations (Córdoba and Alfonso, 2022).

# 3. Methodology:

This research methodology adopts a structured approach to developing a custom environment for space simulation and accompanies AI-driven autonomous spacecraft control systems. The methodology is divided into two primary elements:

1. Development of the custom space simulator environment

This component focuses on the creation of a high-performance, scalable, and deep space simulation system. The key features include:

* CUBA-Accelerated Physics Engine
* Rendering and Visualisation
* Interposes Communication (IPC) Framework

1. AI-driven autonomous spacecraft control systems

The AI will be responsible for decision-making, space navigation, spacecraft management, and anomaly detection. The key element includes:

* Reinforcement Learning (RL) & Transformer-Based Neural Network
* Hierarchical Memory System (STM, IM, LTM)
* Natural Language Processing (NLP) for Mission Command Interpretation
* Anomaly Detection and Spacecraft Health Monitoring
* Context-Aware Mission Execution
* Real-Time Communication with the Simulation and Mission Control

By integrating these components, the research project aims to develop an archetype for a self-sufficient AI autonomous spacecraft control system, reducing reliance on human intervention and enhancing mission autonomy. The following sections will elaborate on the methodology, beginning with the idea being the physics engine :

## 3.1 Development of the custom space simulator environment:

In developing an AI-driven autonomous spacecraft control system, a realistic and computationally efficient simulation environment is needed. This environment should accurately model deep space dynamics, interplanetary navigation, and spacecraft propulsion while supporting real-time decision-making AI. There are existing simulation tools available, such as Orbiter or NASA's GMAT; however, they’re primarily designed for mission analysis and lack the flexibility needed to support real-time AI training and its integration into a spacecraft.  
To solve this problem, a custom space simulation environment was developed that incorporates a high-performance physics engine, a modular architecture, and real-time AI interaction capabilities. The key features of the simulation include:

* A CUBA-accelerated physics engine is needed for the real-time N-body gravitational calculations and the spacecraft’s functions such as propulsion modelling.
* Rendering and visualisation allow verifying the AI actions and system behaviour in the simulation by providing real-time graphical feedback.
* Bidirectional Interposes Communication: A real-time communication with the simulated environment and the spacecraft firmware that enables the AI to receive updated telemetry, issue control commands, and synchronise memory states with the spacecraft system.

In these elements, it allows the simulation to provide an adaptable and scalable framework for testing the AI-driven spacecraft system, anomaly detection, and autonomous mission execution.

The use of a custom physics engine in an AI-driven spacecraft control system can offer several advantages, particularly in optimising the trajectories, enhancing control mechanisms, and system management. One significant benefit is having the ability to simulate complex gravitational interactions and environmental factors, which is important for accurately planning trajectories.

### 3.1.1 CUDA-accelerated physics engine

Due to the complexities and unique challenges associated with simulating space, a custom physics engine was necessary to train an AI-driven autonomous spacecraft control system. Existing physics engines may not be suitable or adequately account for highly nonlinear dynamics and specific spacecraft operational conditions encountered in space, such as renavigation during orbit.

A case study of this was the development of a reinforcement learning algorithm for autonomous docking manoeuvres that highlights the need for a tailored simulation that can accurately model the dynamics of spacecraft in various states, including those that involve rotation targets. So, the proximal policy optimisation demonstrates the importance of having a physics engine that can simulate realistic interactions and constraints to enable robust manoeuvring in uncertain environments while having a low computational cost (Charles et al., 2021).

Moreover, research on autonomous optical-only spacecraft-to-spacecraft tracking emphasises the requirements for having precise modelling of the dynamics involved in the area surrounding Earth. To accurately determine a spacecraft's position using relative measurements from other spacecraft requires a physics engine that can simulate the intricate behaviour of spacecraft under various conditions (Jesse and Daniel, 2023), which is crucial for effective navigation and manoeuvring classification. Standard physics engines are designed for game development or simplified robotics simulation, falling short on the task of modelling large-scale, continuous N-body interactions.

Thus, for this project, a physics engine was developed to meet the specific requirements of having a precision gravitational calculation with real-time propulsion and orbital mechanics and collision handling using continuous collision detection (CCD) at astronomical scales.

#### 3.1.1.1 Modelling Gravitational forces for a multi-Body system

One core responsibility of the physics engine is being able to compute the gravitational influence between multiple celestial bodies. So, for each entity i , the force exerted by another entity j is given by Newton's Law of Universal Gravitation:

A black background with white text

AI-generated content may be incorrect.

Figure 1.1 - Gravitational Forces equation for multiple bodies

Where:

* G is the gravitational constant
* mi, mj are the masses of the entities i and j
* r⃗i, r⃗j are their position vectors of the entities

Computing this for all the pairwise combinations will scale with O(n^2), making it computationally intensive. This shows the need to use GPU-based parallelism.

#### 3.1.1.2 Spacecraft Propulsion and Navigation Control

When it comes to the autonomous navigation, it requires the spacecraft to apply controlled thrust due to environmental variables or mission goals. This means that the net acceleration on the spacecraft is determined by the combined influence of gravitational and propulsion forces divided by the mass of the spacecraft:

A black background with white text

AI-generated content may be incorrect.

Figure 1.2 – Spacecraft Propulsion

Here:

* F gravity is the result of all the gravitational forces acting on the spacecraft
* F thrust is generated by the engine systems
* m is the mass of the spacecraft.

This equation is integrating external dynamics created by the AI idea of the best idea on the needed acceleration at that current step and the internal control, forming the physical interface for AI-driven action selection.

#### 3.1.1.3 Time Integration and Stability

To maintain stability and accuracy, in the physics engine utilises an adaptive integration scheme, such as the Runge-Kutta-Fehlberg (RKF45) method

A black background with white text

AI-generated content may be incorrect.

Figure 1.3 - Explicit Runge–Kutta methods (Wikipedia contributors, 2025)

Where:

* yn is the state of the system at time step n
* h is the adaptive step size
* ki are the intermediate evaluations of the derivative
* bi are the weights corresponding to each stage

The RKF45 allows for the physics engine to adapt it’s time step based on local error estimates, which allows for simulating sudden AI-driven trajectory changes or close gravitational interactions.

#### 3.1.1.4 Simplified Collision Detection

To allow the system to know if a collision has occurred in the environment, a simple collision detection is performed using a continuous collision detection (CCD) model with spherical bounding volumes. Two bodies, A and B, are considered colliding if:

A black background with white text

AI-generated content may be incorrect.

Figure 1.4 – simplified collision detection

Where: rA and rB​ are their respective radii.

To avoid the chance of false positives from fast-moving bodies, relative motion is sampled along the interpolation path between positions over time steps.

#### 3.1.1.5 Justification for CUDA Acceleration

However, the equations described above will demand frequent and large-scale numerical computation, especially the gravitational interactions between multi-body systems, as it grows non-linearly with the number of entities, quickly becoming infeasible for a CPU-bound implementation as the complexity of the simulation increases.

So, adding CUDA acceleration will allow for the parallel execution of computationally intensive routines on the GPU. For tasks such as gravitational force computation, thrust vector integration, collision checks within a bounding volume hierarchy (BVH), and adaptive integration schemes like Runge-Kutta-Fehlberg.

In summary, this custom physics engine accelerated by CUDA will provide the temporal and fidelity necessary for realistic training of an AI in navigation and decision-making in a continuously evolving space environment.

### 3.1.2 Bidirectional Interposes Communication (IPC)

Having efficient coordination between the AI, the spacecraft’s internal management system, and the underlying physics environment requires a robust communication infrastructure. So, to address this requirement, the project implements a bidirectional interposes communication framework to act as a mediation through the spacecraft firmware server, which is based on ZeroMQ over TCP/IP sockets, that allows for asynchronous structured data exchange between independently running subsystems. ensures that there is a modularity and safe real-time responsiveness while maintaining a strict separation of concerns between physical simulation and high-level decision-making.

#### 3.1.2.1 Subsystem Interaction

When it comes to subsystems interaction, it can be separated into three different main systems: the space simulation, the spacecraft firmware server, and the AI-driven autonomous system. The space simulation implements a CUDA-accelerated physics engine, which is responsible for computing the forces acting on the entities, being able to handle large-scale gravitational interactions, collision detection, thrust application, and the evolution of entity states through numerical integration. However, it does not maintain any knowledge of the mission phase, system health, or operational commands.

The spacecraft firmware server functions as the system middleware, bridging the gap between the AI-driven autonomous system and the simulation. The purpose of this subsystem is to keep the spacecraft in an operational state, which includes sensor readings, emergency flags, mission phase, and fuel levels. When the AI commands are received, they are validated and converted into appropriate control instructions. Also, the firmware oversees memory buffers, such as the short-term and intermediate mission memories, which maintain the temporal record of previous state-action-reward sequences.

The AI was developed in Python to form the top layer of the system decision-making by having it interact with the firmware using a ZeroMQ-based IPC channel and making requests for telemetry at a fixed interval. Moreover, after receiving the structured telemetry in a JSON format, the AI will process the data through a transformer-based deep Q-learning model and select an optimal action from its discrete action space. Then the action returned through the IPC layer for the firmware to validate and execute. Below is an example of the telemetry and command transmitted between the AI and the firmware.

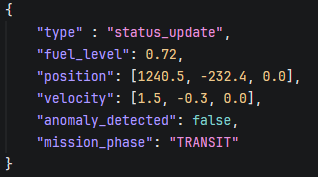


Figure 1.5- Telemetry example (JSON)

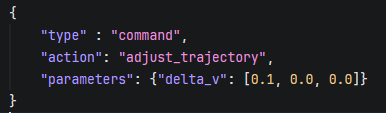


Figure 1.6 - Command example

#### 3.1.2.2 Benefits of the Middleware design

By having a clear separation of the systems with the inclusion of a spacecraft firmware layer, the control architecture helps to produce a more modular and strong system. With the advantage of the middleware design, it allows for command validation by filtering AI-generated control signals through the middleware layer, ensuring that no invalid, unsafe, or physically infeasible actions are passed directly to the physics engine. This architectural pattern mirrors fault-tolerant design strategies in aerospace systems, where mission-critical logic is decoupled from actuator-level control (Truszkowski et al., 2006).

Furthermore, the middleware design integrated a fault isolation and safety handling system through a multi-layered approach. At its core, the system is maintained by a safety manager that continuously maintains parameter monitoring in an isolated thread, preventing control system failures from disabling safety checks.

This system is critical in scenarios where the AI exhibits erratic behaviour or becomes unresponsive; the firmware can independently trigger emergency fallback protocols corresponding with a progressive safety state (NOMINAL, DEGRADED, CRITICAL, EMERGENCY), allowing for a graceful degradation rather than a catastrophic failure. To explain, an emergency protocol happens when critical thresholds are breached, activating a coordinated response like halting thrust, reducing power consumption to 25%, and switching to backup navigation systems. This layered defence mechanism was informed by the lessons learnt from the Earth Observing One Spacecraft, an Autonomous Science Agent by NASA Research Centre (Chien, Sherwood, Tran, Cichy, Rabideau, Castaño, Davies, Mandl, *et al.*, 2005).

3.1.3 Rendering and Visualisation System

One critical aspect of a custom environment for training an autonomous AI system or robotic AI is the ability to verify its interaction with the AI system in the custom environment, seeing how its behaviour and handling of unexpected events through visual feedback. The rendering and visualisation subsystem in the custom environment serves as an observability tool and as a debugging interface, enabling researchers to have the ability to trace the decision-making process of the AI .

#### 3.1.3.1 Motivation for visualisation

The reason for developing a visualisation system in the custom environment was that in deep reinforcement learning systems can be trained through numerical input, but humans would not be able to immediately interpret the data, especially in complex environments where interactions between multiple physical entities, like an N-body simulation, can exhibit emergent behaviour. This means that a visualisation system becomes vital for allowing humans to immediately interpret the AI's and environment's behaviour.

* AI action selection (e.g., thrust adjustments, orbital manoeuvres)
* Gravitational dynamics and orbital trajectories
* Real-time responses to anomaly detection
* Environmental factors such as proximity to hazards

#### 3.1.3.2 System Architecture

The visualisation system is built using OpenGL due to its portability, performance, and flexibility in rendering both 2D elements and full 3D environments. This visual pipeline will receive simulation data through state updates such as positions, velocities, and rotational states directly from the physics engine, allowing it to:

* Draw and animate celestial bodies, spacecraft, and orbit paths
* Overlay contextual data such as fuel status and mission phase via graphical HUDs

#### 3.1.3.3 Synchronisation with AI and IPC

Even though the rendering engine is separate from the AI system, the engine can reflect the outcome of the AI decisions in the virtual environment, like, for example, when the AI chooses to issue an adjust trajectory command through the IPC, resulting in the state of the spacecraft being altered through a course correction or orientation change that can be seen in the rendered scene.

#### 3.1.3.4 Benefits of Real-Time Visual Feedback

There are several benefits offered in implementing a visualisation system. The system enables the ability for developers to confirm whether AI action is producing the intended effects. It also improves the developer's ability to understand complex interactions such as multibody gravitational slingshots and orbit transfers. Additionally, it serves as a valuable debugging tool for analysing unexpected behaviours or catastrophic system failures.

## 3.2 AI-driven autonomous spacecraft control systems:

In parallel with the custom simulation, this research implements an AI-driven control system designed to govern the behaviour of a spacecraft autonomously through context-sensitive decision-making that evolves over time. Unlike traditional systems that are dependent on human operators and pre-programmed instruments. The AI requires an intelligent framework that can interpret sensor data, manage mission execution in real time, and adapt to uncertain environments.

To meet these requirements, the AI architecture integrates deep learning, cognitive-inspired memory management, natural language interpretation, and real-time feedback mechanisms. The key features of the control system include:

* Transformer-based Reinforcement Learning Model: This component enables the AI to learn optimal control strategies through interaction with the environment. This architecture allows the AI to reason over sequences of sensor data to support better pattern recognition and stability of the decision process.
* Hierarchical Memory System: The AI incorporated a multi-tiered memory system inspired by human cognition:
  + Short-term Memory (STM)
  + Intermediate Memory (IM)
  + Long-term Memory
* Anomaly Detection System: A neural autoencoder-based anomaly detector system identifies deviation from the normal spacecraft behaviour, allowing the AI to trigger contingency actions.

* Natural Language Command Interpretation: The AI system can parse and interpret structured and semi-structured mission directives through a lightweight NLP pipeline, enabling mission communication to use flexible language inputs.
* Contextual Mission Phase Awareness: The AI modifies its reasoning based on predefined mission phases, adjusting its decision and control parameters.

Together, these architectural components provide the cognitive and operational capabilities needed for the AI to function as a fully autonomous agent. The system is capable of executing deep-space missions, responding to anomalies, and continuously learning from experience.

### 3.2.1 Transformer-Based Reinforcement Learning Model:

The core of the AI is a transformer-based reinforcement learning model that was designed to support temporal reasoning, autonomous control, and decision-making in complex and evolving space environments. It’s operating as a Q-learning agent, where a neural network is trained to approximate Q-values, the expected cumulative reward across a discrete action space given a history of state inputs. These Q-values form the foundation of the agent’s policy for action selection.

Traditional reinforcement learning (RL) methods will often use shallow feedforward architectures or recurrent neural networks (RNNs) to process temporal dependencies. However, these architectures struggle in environments characterised by long-range temporal dependencies, partial observability, and sparse reward signals, all of which are prevalent in space navigation and anomaly response scenarios. To overcome these limitations, a transformer-based neural network originally proposed by Vaswani et al. (2017) which leverages a multi-head self-attention mechanism that allows the model to learn relationships between all elements of an input sequence in parallel. In this context the control system enables the AI to weigh the recent state transitions, anomalies, and mission data across time, allowing for the improvement of the temporal reasoning and action selection of the AI.

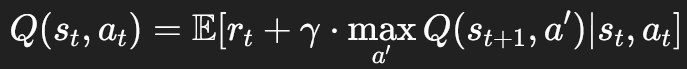
The reinforcement learning paradigm is framed around a Markov Decision Process (MDP), where the agent interacts with the environment at discrete time steps t by observing a state st, taking an action at​, and receiving a reward rt​. The transformer processes a sequence of recent states (or embeddings thereof). St−n:t and outputs Q-values for each possible action:

Figure 1.7 – Transformer

The key advantages of using a transformer in this setting include:

* Temporal abstraction: The model learns to prioritise events or observations with long-term implications.
* Scalability: Attention-based models scale better with long input sequences than RNNs or LSTMs.
* Multi-modal integration: The model can incorporate many kinds of sensor data, like velocity vectors, anomaly flags, and position coordinates.

In this system, the transformer is trained using experience replay sampled from intermediate memory (IM), with target Q-values computed using a delayed copy of the network to stabilise learning. The loss function is based on the mean squared error between predicted and target Q-values, optimised via Adam.

Precedents in using transformer architectures for RL are seen in works like Decision Transformer (Chen et al., 2021) and Trajectory Transformer (Janner et al., 2021), which validate the use of sequence modelling for control tasks. This project extends those principles into a real-time aerospace simulation context with feedback-driven action execution and persistent memory integration.

Exploration Strategy

The system employs an ε-greedy strategy to balance exploration and exploitation:

* Exploitation happens with probability 1 - ϵ, where the model selects the actions with the highest predicted Q-value.
* Exploration happens at the beginning of the model with probability ϵ , selecting random actions to encourage behavioural discovery and diversity of alternative strategies.

The exploration rate ϵ will decay over time, beginning at 1.0 and gradually reducing to the minimum threshold of 0.01, allowing the model to transition from a stochastic exploration to a stable policy as it gains confidence in its learning.

### 3.2.2 Hierarchical memory system

When it comes to autonomous systems operating in high-risk environments, a single-layer memory architecture is insufficient for sustained reasoning, learning, and adaptation due to the large amount of data generated by the environment. To address this, the AI system employs a hierarchical memory system inspired by cognitive architectures in neuroscience and artificial intelligence. The system is composed of three distinct, interacting memory layers:

* Short-Term Memory (STM)
* Intermediate Memory (IM)
* Long-Term Memory (LTM)

Each layer serves a specialised function in supporting the AI reactive control, strategic mission planning, and learning from experiences.

#### 3.2.2.1 Short-Term Memory (STM)

The STM stores recent state observations and environmental context relevant for immediate decision-making. Implemented as a circular buffer in the spacecraft firmware, providing fast in-memory access to the last n sensor states, mission conditions, and AI actions. Enabling the transformer-based model to form sequences for temporal attention during each mission step.

STM allows the AI agent to react quickly to anomalies or changes in the environment, emulating how biological organisms maintain immediate awareness (Baddeley, 1992). Having a limited horizon prevents memory saturation due to unnecessary data and ensures only contextually relevant data is processed.

#### 3.2.2.2 Intermediate Memory (IM)

The IM acts as a bridge between the mechanisms STM and LTM, buffering the state-action-reward sequences across episodes. This buffer supports:

* Experience Replay in Q-learning
* Temporal Pattern Capture (e.g., trajectory drift, recurring anomalies)
* Memory Consolidation triggers, wherein significant events are transferred to LTM

This module is implemented in Python and operates asynchronously, collecting episodic data from the simulation and tagging them with mission phase metadata.

This structure is meant to reflect works on episodic memory in AI, such as that seen in DeepMind’s Differentiable Neural Computer, which proposes a memory matrix for adaptive task switching and memory retention (Graves et al., 2016).

#### 3.2.2.4 Long-Term Memory (LTM)

While STM and IM are handled by the spacecraft firmware and have a local runtime, LTM resides in the AI system’s high-level decision module implemented via a transformer encoder-decoder architecture from TensorFlow. The LTM stores compressed mission strategies, mission insights, learnt policies, and anomaly trajectories.

The LTM is updated periodically based on reward gradients or on heuristic rules like mission phase transitions or anomaly resolution. The LTM functions similarly to the memory-based meta-learning approaches that allow the AI agents to generalise across tasks and encode transferable knowledge (Ritter et al., 2018)

#### 3.2.2.5 Memory Integration Workflow

* STM captures immediate input: executed action, sensor snapshots, transient conditions.
* IM logs the episodic sequences for replay and training.
* LTM periodically updated from the IM based on a consolidation thresholds like mission-critical event markers, prolonged anomalies, or reward spikes.
* The Transformer model will then consume the sequences from STM and IM to guide real-time decisions, while LTM influences model updates and strategic reasoning.

#### 3.2.2.6 Rationale and Contributions

The hierarchical memory architecture provides cognitive depth to the AI, enabling it to react, reason, and reflect on its decision-making process while ensuring that there is both stability and adaptability through the separations of transient information from persistent knowledge. Additionally, the architecture enhances the learning by integrating real-time stimuli from past episodic and strategic experiences.

### 3.2.3 Anomaly Detection System

In the autonomous spacecraft systems, anomaly detection is critical for a mission's success, as it enables early identification of deviations from nominal behaviour that could compromise the spacecraft's integrity or the mission. Due to the complexity and uncertainty of deep space, a traditional rules-based fault detection system is insufficient. So instead, the anomaly detection system utilises a machine learning-based system that provides dynamic, adaptive monitoring of spacecraft health and environmental conditions.

#### 3.2.3.1 Motivation for the autonomous anomaly detection

The reason for the anomaly detection system was because human monitoring becomes impractical during deep space missions due to significant communication delays. So, this means that the AI must independently identify the hazardous conditions and initiate protective actions or corrective actions without waiting for human intervention. The type of anomalies may include:

* Unexpected environmental hazards
* System degradation like fuel leakages and thruster malfunctions
* Sensor drift or failure
* Mission deviations like orbital instability

#### 3.2.3.2 Anomaly Detection Architecture

The core of the detection mechanism is based on an autoencoder, which is a special type of unsupervised feedforward neural network trained to reconstruct normal operational data (Sakurada and Yairi, 2014). An autoencoder works by learning to compress input telemetry into a latent space and then reconstruct it. Anomalies are detected when the reconstructed latent space errors exceed a learnt threshold.

Then the reconstruction error E can be defined as:

A black and white text with a white x

AI-generated content may be incorrect.

Figure 1.8 - reconstruction error

Where:

* X denoted the input telemetry vector like sensor readings or system status
* X^ is the autoencoder's reconstructed output

And an anomaly is flagged if :

A white text on a black background

AI-generated content may be incorrect.

Figure 1.9 – anomaly flag

Where ϵthreshold is a context-sensitive threshold that adjusted based on mission phase and environmental conditions.

#### 3.2.3.3 Training and deployment:

* Training phase: The autoencoders are trained offline using a dataset generated by the simulation environment .
* Online monitoring phase: During live operations, incoming telemetry data is continuously passed through the trained autoencoder, and reconstruction errors are computed in real time and flagged

Anomalies are classified as low-severity like minor sensor drift which trigger a caution alerts, medium-severity like fuel leak detection will trigger pre-emptive countermeasures, and high-severity like loss of control authority triggers an emergency protocols and mission re-evaluation.

#### 3.2.3.4 Integration with the AI control Loop

By embedding the anomaly detection system within the AI control loop, the spacecraft can dynamically adapt decision strategies, shift mission phases, and prioritise emergency protocol.

A diagram of a process flow

AI-generated content may be incorrect.

Figure 1.10 - Anomaly Detection System integration

### 3.2.4 Natural Language Command Interpretation

Beyond the real-time communication capabilities of the autonomous system, the ability to interpret complex mission instructions issued in natural language becomes critical. Rather than relying solely on a predefined instructions format, the AI is created to understand, parse, and prioritise semi-structured or unstructured natural language inputs originating from Earth. This capability enables the AI to have greater flexibility in mission execution, reducing the operational burden on humans crews, and allow for dynamic re-tasking during missions.

#### 3.2.4.1 Motivation for Natural Language understanding:

In traditional spacecraft command systems, dependence is high on structured binary sequences. While this ensures their precision, it severely limits operational flexibility and responsiveness to unforeseen events. Furthermore, in long-duration missions where human oversight is minimal, the ability to transmit intent rather than micromanaged sequences becomes increasingly valuable.

So, the natural language-based command interfaces will need to support:

* Dynamic mission adjustments without the need for reprogramming low level behaviours
* Priority updates based on the emergent of hazards or mission opportunities.

#### 3.2.4.2 System Architecture

The natural language system consists of several layers:

1. The command reception layer takes incoming text-based commands via the Interposes Communication (IPC) framework using ZeroMQ.
2. Parsing and Tokenization pre-processed the command through a standard natural language processing (NLP) technique, including tokenization, stemming, and stop-word removal using libraries such as NLTK (Bird, Klein and Loper, 2009).
3. Command classification happens with a lightweight neural classifier or a keyword matching system that maps natural language commands into predefined operational categories, such as Adjust Trajectory, Investigate Anomaly, and others. For more complex instruction, sequence-to-sequence models like a transformer encoder can be applied to the map phrases to the action plans.
4. An intent prioritisation module will be used if multiple commands are received or if a command conflicts with ongoing mission phases, where the AI uses a priority arbitration mechanism based on mission phase, risk assessment, and reward projections.
5. An action translation layer existed for translating parsed command into one or more executable actions, aligning with the available action space of the AI.

A diagram of a system

AI-generated content may be incorrect.

Figure 1.11 - Natural language system architecture

An example workflow of the natural language process would be that mission control sends a message to the AI saying, "Prioritise anomaly investigation and reduce velocity to conserve fuel." The system would take the text and parse it into key intents like Investigate anomaly → investigate anomaly and Conserve fuel → decrease velocity. Then commands are classified and prioritised, while immediate actions are triggered if needed, and finally, the transformer model’s reward function is slightly modified to favour fuel-saving behaviours.

#### 3.2.4.3 Integration with the Reinforcement Learning Agent

Natural language commands are fed into the agent either as an external policy override, where parsed actions are immediately influencing decision-making, or as high-priority inputs that modify the reward function or the bias of the Transformer’s attention mechanism toward mission goals.

This dual-path integration ensures that while the autonomy is preserved, human adjustments can be respected without destabilising the learning process.

### 3.2.5 Contextual Mission Phase Awareness

In deep-space operations, the priorities and dynamics of spacecraft change significantly across different mission phases, so a one-size-fits-all decision-making policy is insufficient for long-term autonomy. To address this, the AI incorporates a contextual mission phase awareness, enabling it to have the ability to adapt its behaviour, priorities, and risk thresholds according to the current operational context.

#### 3.2.5.1 Motivation for Phase-Based Adaptation

When it comes to spacecraft missions, they can typically separate into discrete phases, each associated with unique operational aims and risk profiles:

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | AI behaviour | Reward shaping | Anomaly sensitivity |
| Launch | Critical thrust management, and trajectory alignment | Penalise high acceleration variances | High |
| Transit | Long-duration cruise control, resource conservation, and anomaly monitoring | Reward efficient cruising | Medium |
| Exploration | surface proximity manoeuvres, intensive data collection, and environmental hazard response | Reward data collection and proximity manoeuvres | High |
| Critical | handle emergent anomalies, course corrections, and survival manoeuvres | Heavy penalty for inaction | Very high |
| Emergency | system recovery, orbit stabling, and fail-safe execution | Reward system stabilisation | Critical |

The list shows that different phases demand different strategic focuses, like, for example, risk tolerance during the launch phase must be minimal, whereas during the exploration phase, flexibility and hazard adaptation are prioritised.

So that means embedding mission phase awareness will allow the AI to dynamically reweight reward signals, adjust action selection policies, modify anomaly sensitivity thresholds, and alter exploration and exploitation balances.

#### 3.2.5.2 System Architecture

The mission phase awareness system architecture can be broken down into three key components: The core component of the system is that it can track the current mission phase, like launch, orbit, etc., with confidence values and updates based on sensor readings and mission parameters. Additionally, it integrates with the AI data manager and adapts its state based on environmental conditions, thereby maintaining context through phase transitions.   
The system also provides phase-specific policies that influence the AI decision-making process through the modification of the reward tailored to each phase. These modifications adjust how the system evaluates actions, such as prioritising fuel efficiency, speed, or safety depending on the mission context. Allowing the AI to adapt its behaviour appropriately as mission conditions change.

# 4. Implementation overview

The implementation phase transforms the methodological design and theories outlined previously into a practical model enabling spacecraft to navigate, make decisions, and handle problems in a simulated environment. This implementation serves as an open framework within which an AI agent can operate, interact, and learn over time.

The system architecture employs a modular design that spans multiple programming languages and paradigms, such as the high-performance physics and environmental simulation was implemented in C++ due to the performance benefits, and the deep learning, most of the memory system, and agentic logic were implemented through Python. While interacting with each other via a bidirectional interprocess communication (IPC) mechanism, mediated by a firmware layer that ensures temporal decoupling and real-time data consistency.

The key goal for this implementation includes ensuring that the simulation and the AI can process data within a bound latency threshold. While having strict modular separation between the AI logic, physics engine, and the memory system to maximise testability and scalability. Also, the system should be able to produce comprehensive logging and state introspection features for debugging and post-mission analysis.

The implementation will proceed in five core modules:

1. The physics and environmental simulation engine (C++ / CUDA) which implements the realistic gravitational modelling, spacecraft model, and simple collision detection.
2. The AI agent layer (Python) will present the implementation of the Transformer-based AI with a memory hierarchy and hierarchical control logic.
3. The bidirectional interposes communication (ZeroMQ) handles asynchronous messaging between simulation and AI through the firmware application.
4. The Monitoring and visualisation system (OpenGL + Logging) for providing real-time render feedback and detailed introspection of the AI actions.

This section will deal with the implementation of each component, demonstrating how the whole system architecture works together to realise the key goals of the project. It also needed to be noted that this section would not be showing all the code due to it being over a thousand lines of code.

The key library and framework used in the system where:

For the python environment:

* TensorFlow(v2.9.1) for the transformer-based neural network architecture
* NumPy (v1.22.4) for efficient numerical computations and array manipulations
* pyzmq (v24.0.1) for interposes communication between the AI and simulation components

For the C++ environment:

* CUDA Toolkit (v12.7) for GPU-accelerated physics calculations
* Cppzmq (v1.0.2) for interposes communication

## 4.1 The AI agent layer (Python)

The AI agent layer is the cognitive brain of the autonomous spacecraft systems, responsible for orchestrating the strategic decision-making, temporal reasoning, anomaly response, and trajectory planning. The AI layer is primarily implemented in Python and integrates multiple subsystems, including the transformer-based reinforcement learning engine, hierarchical memory modules, and interposes communication interfaces to coordinate with various subsystems.

Within the agent layer, the main.py file hosts the AICaptain class, which serves as the primary controller for the autonomous system. This class continuously evaluates mission objectives, fuel levels, and sensor inputs, balancing the demands of mission progress against fuel and safety constraints. For instance, in an EMERGENCY, the AICaptain prioritises immediate resolution of anomalies.

### 4.1.1 Transformer-Based Architecture

At the heart of AI is a Transformer-based RL model, a neural network designed for sequential decision-making in a space environment. The model is trained to output Q-values for discrete action choices based on a temporal sequence of sensor states. The model architecture is composed of:

* A 6-transformer encoder layers, each with 8 self-attention heads.
* Layer normalisation, residual connections, and dropout for regularisation
* A global average pooling layer that aggregates the sequence-level features
* A multi-head output layer that comprises a Q-value prediction for 7 discrete actions such as maintain\_course, adjust\_trajectory, increase\_velocity, etc., and a parametric output for actions requiring continuous control like trajectory\_angles, and delta\_v.

The design of the transformer is a multi-output that allows for the model not only to determine what action to take but also how to execute it with precise parameterisation. Which is important for spacecraft navigation and anomaly response that require continuous control. Part of the transformer-based model can be seen in figure 2.1, showing how it was implemented in Python with its multi-output design.

**A computer screen shot of a program code

AI-generated content may be incorrect.**

Figure 2.1 – Transformer basic architecture

### 4.1.2 Dynamic Contextual Reasoning via Sequence Length Adaptation

To enable the ability for the AI to support mission-phase-specific behaviour, a subclass of the transformer model called DynamicTransformerModel was made that dynamically adjusts the sequence length of input histories based on the current mission state, such as Launch Phase: 5 steps will always favour rapid decision-making; Transit Phase: 15 steps will have balanced responsiveness with context; Exploration Phase: 25 steps will leverage extended memory; and in Emergency Phase: 3 steps will want to have a prioritised immediate reaction.

This design was inspired by the adaptive attention models in context-sensitive domains that allow the agent to weigh the temporal dependencies differently depending on the current operational constraints (Vaswani et al., 2017).

### 4.1.3 Inference and Action Execution pipeline

The pipeline that handles the inference and action executions can be represented in an operational loop through the AI captain, which interprets its environment, initiates commands, and selects context-aware actions. This logic is tightly coupled with the mission phase logic, sequence management, and parametric control system in the AI.

#### 4.1.3.1 Input state sequence

At each of the AI decision steps, it will receive a state\_sequence, a temporally ordered series of structured observations derived from spacecraft sensors. Each of the elements in the sequence are encoded with key operational parameters such as position, velocity, anomaly status, and fuel levels. This sequence will form the basis for the input to the transformer-based inference system.

4.1.3.2 Sequence Adjustment

Then the DynamicTransformerModel adapts the temporal window based on the current mission phase. Like, for example, during a launch, a shorter context window will be better due to supporting rapid decisions, while exploration phases expand to 25 steps to accommodate long-horizon planning. However, if the state sequence exceeds the required length, the most recent subset is extracted. If it is shorter, zero-padding is applied to preserve positional encoding and sequence dimensionality.

#### 4.1.3.2 Prediction

The adjusted sequence is passed through the dynamic transformer to output two predictive streams. One of the streams is the Q-values for each action in the discrete policy space, and the other is the parametric output for actions requiring continuous control. These outputs are normalised during training and denormalised during inference using mission-specific parameter bounds. Also, in figure 2.2 it shows how the AI predication is implemented in DynamicTransformerModel.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 2.2 – Transformer Model prediction

#### 4.1.3.3 Action Selection

An ε-greedy policy is applied to determine the action, and it is found in the get\_action() method of the AI caption class. During the explorationa random number is generated using the NumPy random method, and if that random number is less than self. epsilon, the system selects a random action. While in the exploitation, the action with the highest predicted Q-value is selected. This only happens if the random number is greater than or equal to self. epsilon, leading to the system using the model to predict Q-values for the current state sequence, and having the action with the highest Q-value selected. This action selection logic can be seen in figure 2.3.

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 2.3 – Action selection logic

#### 4.1.3.4 Action Execution

The action and its parameters are transmitted via IPC to the firmware controller, which applies the corresponding command to the spacecraft simulation. For example, to adjust trajectory will apply thrust along a predicted vector with specified magnitude and orientation, and to increase velocity will add a ∆v adjustment proportional to the predicted output.

After execution of the action in the environment, it evolves according to physical dynamics and updates all relevant entity states, then the new state is appended to the AI's buffer, forming the next state\_sequence for the following inference cycle. This closes the decision loop and initiates the next iteration of planning.

### 4.1.4 Training Regime

The AI training regime is grounded in RL, leveraging a transformer-based architecture and experience replay to learn action policies within a dynamic and hazardous space environment. The training process is conducted online, during simulation runtime.

#### 4.1.4.1 Experience Collection and Memory Management

Experience is gathered continuously as the AI executes its mission while operating the spacecraft. Each interaction with the environment generates a 5-tuple:  
(st, at, rt, st+1, dt), which represents the state, action, reward, next state, and done signal.

The experiences are stored in the mission memory buffer, a high-capacity replay structure implemented as a deque. The buffer enables off-policy learning by breaking the temporal correlations between experiences and supporting minibatch sampling during training updates.

#### 4.1.4.2 Training Architecture

The training process is performed using the train() method in the AI captain class, which takes samples of minibatches from the memory and computes the Q-learning target. The core of the loop works by first sampling batches of experiences from memory, then formatting the sequences using historical context windows while forward passing through the transformer model and calculating the Q-values and the loss between predicted and target Q-values and finally updating the model via backpropagation.

Also, the AI reward function is modular and context sensitive. This means that the AI can penalise unnecessary thrusts or suboptimal manoeuvres if it is ordered to focus on fuel efficiency or be rewarded for completing tasks within fewer simulation ticks.

#### 4.1.4.3 Phase-Aware Scheduling

The training is invoked periodically during the mission execution, and the system employs a phase-aware schedule that invokes training every 10 steps while also having specialised subsystems such as the Anomaly Detector and Hazard Detector retrained using recent sensor data contextualised by the current mission phase. These detectors enhance policy shaping by embedding situational awareness into the model’s reward structure and inputs.

## 4.2 Interposes Middleware (ZeroMQ)

In this distributed architecture where the physics simulation engine, spacecraft firmware, and the AI control module run as independently managed processes, a robust and low-latency communication layer is vital for the architecture. To address these requirements, the system employs a bidirectional interposes communication (IPC) framework in the spacecraft firmware using ZeroMQ, a high-performance asynchronous messaging library known for its lightweight footprint and versatility in concurrent systems. The layer is responsible for message mediation, telemetry exchange, and safe access to shared memory and control functions.

### 4.2.1 Layered Architecture

The middleware is designed with a modular layered architecture, where each of the software components assumes a distinct functional role. These separations allow for improved maintainability, testing, and fault isolation. Two primary layers are critical:

The Spacecraft Firmware Layer is implemented in the spacecraftServer.cpp, a middleware server. Its job is to listen for inbound requests, process messages, invoke simulation logic, and respond with structured telemetry or control acknowledgements. All messages are encoded in JSON, providing extensible, human-readable communication across process and language boundaries.

The memory management layer implemented in the IntermediateMemory.h file manages the concurrent read and write access to the mission data while also implementing a thread-safe circular buffer guarded by mutexes and atomic primitives to ensure that multiple threads can write to or retrieve from the memory without race conditions or data corruption.

This division allows for the AI that’s running in Python to operate independently from the simulation internals, allowing it to focus on high-level decision-making while delegating sensor and actuator access to the middleware.

#### 4.2.1.1 Messaging Pattern

The architecture uses a request-reply pattern over ZeroMQ where the AI (client) sends a sensor query or an action command to the middleware where the spacecraft firmware server takes the JSON message and parses it while logging and validating the request in the system. Then, once the request is validated, it will trigger simulation logic or query the memory, and finally it serialises the response and sends it back to the AI agent.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 2.4 - JSON message schema

In figure 2.4 it shows how each JSON message follows a standardised schema. By having a standardised schema across subsystems, it will support extensibility and imperceptibility across simulation modules.

#### 4.2.1.2 Concurrency and Fault Tolerance

To ensure that the reliability and responsiveness of the middleware under real-time conditions required both robust concurrency mechanisms and fault tolerance strategies. The system does this by incorporating multithreading, mutual exclusion, and checkpoint-based recovery to ensure operational integrity, especially during critical mission phases.

To support concurrent operations across subsystems, the system uses C++ standard concurrency primitives:

* Threaded monitoring in the safetyManager.cpp with a dedicated monitorThread that continuously runs a safety diagnostics loop in parallel with other operations. This enables uninterrupted health checks on thermal, power, and subsystem responsiveness metrics without blocking control or memory logic.
* Mutex-protected memory access in the IntermediateMemory.cpp where all memory access is synchronised via a std::mutex, guaranteeing that their thread-safe interaction with the memory buffer ensure no two threads simultaneously corrupt or read inconsistent data.

When it comes to fault tolerance, the systems employ a redundant strategy to ensure graceful degradation and recovery in case of systemic faults.

* One of the fault tolerances is a checkpoint for memory recovery which happens before each write operation; the createCheckpoint() function captures a snapshot of the memory buffer along with a checksum. If the data were to get corrupted or an exception is raised, the restoreFromCheckpoint() function reverts the system to the last known good state. This rollback mechanism limits data loss and maintains internal consistency during anomalies.

The concurrency and fault recovery of the middleware system is crucial for maintaining robustness in autonomous, AI-integrated spaceflight software systems, particularly under the uncertain, asynchronous, and resource-constrained environments typical of deep-space missions.

## 4.3 Physics and Environmental Simulation Engine (C++/CUDA)

The core of the spacecraft simulation environment would be the physics and environmental engine implemented in C++ with GPU acceleration via the CUDA toolkit. This engine governs the evolutions of the physics state in the space environment, including the management of the gravitational interactions, spacecraft manoeuvring, and collision events. While being tightly integrated with the ECS framework to enable the modular and scalable state updates across many entities.

### 4.3.1 Architecture and core design

The physics engine is designed to be a modular object-orientated simulation system that is optimised for high-performance real-time space dynamics. The design prioritises separation of concerns, predictable control flow, and hardware-level concurrency, allowing each subsystem to function independently while integrating with the AI agent and mission control interfaces.

At the core of this engine is the PhysicsEngine class, which encapsulates all major stages of the update loop, including initialisation, data synchronisation, kernel launching, and cleanup. In the simulation, each object is defined through an explicit body or spaceship class with properties such as mass, position, velocity, acceleration, and radius. These classes are designed for GPU compatibility using POD (Plain Old Data) layouts and are optimised for direct transfer between host and device memory.

### 4.3.2 Object-Oriented Layering

The simulation is divided into a set of modules, like the body module, which define the position, velocity, and mass of objects within the simulation. Meaning that every dynamic object derives from this structure and is subject to gravitational influence, ensuring realistic motion and interactions.

Another module would be the spaceship, which inherits from the body module, adding control properties such as thrust magnitude, fuel reserves, and orientation. This allows spacecraft to function within the same gravitational system as passive bodies while also responding to AI-driven control vectors.

Then there is the trajectory module that stores the historical positional data of each object, enabling post-analysis and visual validation by maintaining a record of movement over time. This module is meant to support accurate assessments of object trajectories.

Lastly, there's the PhysicsEngine Module that manages all memory buffers, launches CUDA kernels for dynamics calculations, and handles synchronisation with external subsystems. This module ensures their efficient computation and seamless integration within the broader simulation framework.

Each of these modules is independently testable and loosely coupled with each other, following the SOLID principles for architectural design, improving maintainability and debugging while also supporting future extensibility, such as the introduction of new force models.

### 4.3.3 GPU-Accelerated Physical Behaviours and Dynamics

Achieving the realism in the simulated environment is done through the GPU-accelerated computation of gravitational dynamics, spacecraft propulsion, orbital integration, collision detection, and trajectory logging. These systems are implemented in the PhysicsEngine using a suite of CUDA kernels and device-side routines that operate in parallel across simulation entities to enable the system to maintain real-time performance in a computationally demanding N-body context.

This section is going to outline each major behaviour that is encapsulated into the CUDA-executed algorithms.

#### 4.3.3.1 Gravitational Force Accumulation

When it comes to the gravitational interactions between celestial bodies, they can be computed with Newton's Law of Universal Gravitation, which can be seen in Figure 1.1; however, when computing, there is a need to add a softening factor to the equation to be like the one seen in Figure 2.5:

A black background with white text

AI-generated content may be incorrect.

Figure 2.5 - gravitational interactions equation with softening

Here, ϵ is a representation of the softening factor, which is set to 1.0 X 10^9 km to prevent numerical divergence during close-range interactions. This function is implemented in the computeDerivatives kernel and executed on the GPU for every pair body, and part of the code can be seen in figure 2.6:

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 2.6 – Compute derivatives

For the simulation with a high object count, the engine utilises the Barnes-Hut approximation to reduce computational complexity from O(n^2) to O(n log n). This is achieved by spatially organising the simulation into an octree data structure and approximating distant clusters of bodies as a single mass node. The GPU-accelerated kernel computeBarnesHutForcesKernel traverses the octree and applies aggregated forces accordingly.

#### 4.3.3.2 RKF45 Integration Scheme

Using a 6-stage Runge-Kutta-Fehlberg (RKF45) integrator guarantees that there is an accurate and steady motion prediction, and this crucially, for resolving tightly curved trajectories or fast dynamics like thruster burns and gravitational assists, this technique allows adaptive time stepping.

Where each stage of the RKF45 is computed using a dedicated CUDA kernel, in figure 2.7 it shows stage 4 of the RKF45.

A computer screen shot of a code

AI-generated content may be incorrect.

Figure 2.7 – Stage 4 of the RKF45

Where the final state is computed as a weighted sum of the intermediate derivative evaluations, and the equation for calculating the weighted can be seen in figure 1.3. Where k⃗i​ represents the state derivative at stage i, and bi​ are precomputed RKF weights to produce a balanced accuracy and performance, and its GPU implementation allows for parallel updates of hundreds of bodies per frame.

#### 4.3.3.3 Collision Detection and Response

In the simulation, it implements a soft-sphere collision detection using Euclidean proximity for each body pair (i,j); a collision is flagged if the radius added together is less than the radius mins for each pair, and the equation can be seen in figure 1.4. The logic is encapsulated in detectCollisionsKernel and parallelised over the set of active bodies; the implementation can be seen in figure 2.8.

A computer screen shot of code

AI-generated content may be incorrect.

Figure 2.8 – Detect Collisions Kernel

#### 4.3.3.4 Trajectory Logging and Analysis

To enable the ability for analysis and visualisation, each body's motion and history are captured via a fixed-length circular buffer, implemented in recordTrajectoryPoints. This data supports both post-mission review and real-time feedback to the AI. The key feature is that it is logged at a fixed interval, the oldest points are overwritten once buffer capacity is reached, and all trajectory arrays are pre-allocated in device memory. In figure 2.9 it shows the implementation of the recordTrajectoryPoints function.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 2.9

The trajectory system is useful for generating orbit visualisations and analysis the behaviour of the system.

## 4.4 Monitoring and Visualisation System (OpenGL + Logging)

Developing and evaluating the autonomous control system requires the ability to track AI behaviour and environmental dynamics through a dual-pronged approach of having a basic OpenGL rendering engine for graphical feedback and trajectory analysis and a timestamped logging and event inspection framework for behavioural traceability and anomaly debugging. Enabling developers and researchers to observe, verify, and validate how the AI interacts with the physics-based simulation.

### 4.4.1 OpenGL-Based Visualisation System

The visualisation subsystem renders the simulated space environment using OpenGL, providing graphical feedback on the various elements, including spacecraft and celestial body trajectories and thrust directions. This rendering system enhances the understanding of dynamic interactions within the simulation.

The rendering pipeline of the systems employs several key components to achieve efficient and accurate rendering. Vertex Buffers store all entity positions (bodies, ships) on the GPU side, ensuring fast access and processing, and a trajectorypath is visualised using polylines derived from the historical position data stored in the trajectory in a logging buffer for post-analysis of object motion.

This interface is meant to serve as a feedback loop for observing AI learning progress, control accuracy, and physical interactions.

### 4.4.2 Logging and Introspection Framework

In parallel to the visualisation system, a structured logging framework records the system activity at multiple verbosity levels, including INFO, DEBUG, WARNING, and ERROR, playing the crucial role of maintaining system integrity by helping to diagnose AI control decisions and identify faults or anomalies in the system.

#### 4.4.2.1 Logging architecture

The structured logging framework is structured to capture a wide range of operational data to facilitate debugging, optimisation, and post-mortem analysis. One type of data is action that records each action selected by the AI, including Q-values, parameters, and mission context. Another type is error tracking that captures stack traces or exceptions raised during AI inference or memory faults. Also, there's an event tracking that tracks significant occurrences such as command reception, anomaly detection, and fuel depletion, providing a high-level overview of system interactions, making it easier to identify critical events within the simulation.



Figure 2.10 – example log entry

In figure 2.10 it shows two examples of log entries that highlight the AI's failure to complete an action and where it succeeds in completing an action. The logging framework stores entries like Figure 2.10 in a structured .log file, ensuring that their organisation is easy to access.

## 4..5 Implementation Overview – Summary

The implementation of the Space Traveller AI system embodies an integrated architecture of modular components, each designed to simulate, control, and observe deep-space autonomous operations. From the CUDA-accelerated physics engine to the AI agent layer, it produces an extensible platform for simulating and testing intelligent autonomous agents in space mission scenarios, combining deep learning, high-performance computing, and fault-tolerant systems engineering.

# 5. Results and Discussion

This section discusses the experimental findings and performance validation of the modular components that comprise the AI-driven autonomous control system. Each system was developed and evaluated as a standalone project to ensure functional correctness, stability, and performance within its respective domain before being integrated into the larger autonomous space framework.

Once individual validation of each subsystem was completed, they were integrated together via a ZeroMQ-based interposes communication framework to evaluate their interactions. The final testing focused on verifying end-to-end behaviour through telemetry exchange, decision-making coherence, and system responsiveness.

## 5.1 Evaluation Criteria

To assess the effectiveness of the AI-driven autonomous system, following these evaluation criteria

|  |  |
| --- | --- |
| Subsystem | Evaluation Criteria |
| Space Environment | -Accuracy of gravitational simulations (Newtonian & Barnes-Hut)  -Stability of RKF45 integration  -GPU computation time & performance |
| AI Agent | -Mission success rate (goal completion)  -Decision latency (response to new input)  -Policy convergence in controlled training  -Memory system effectiveness (LTM interactions) |
| Spacecraft Firmware | -Command reliability  -Fault detection and recovery  -Telemetry buffering and synchronisation accuracy |
| Integrated System | -Real-time responsiveness  -Interoperability via IPC (message passing success/failure rate)  -Autonomous decision alignment with mission goals |

## 5.2 Space Environment Simulation Evaluation

The validation for the space environment simulation was done through a series of structured tests focused on physical accuracy, numerical stability, and computational performance under GPU acceleration using the Google Test Library. The simulation supports both Newtonian and Barnes-Hut gravitational models and uses an adaptive RKF45 integrator for accurate and stable time evolution of orbital systems.

### 5.2.1 Accuracy and Stability of Newtonian Gravity

The Newtonian gravitational implementation presents exceptional precision. In a full-orbit simulation of a 9-body solar system with a 1-day timestep over one Earth year, because energy conservation was exact, with 0% deviation across the simulation period and orbital fidelity was maintained, with Earth returning precisely to its starting position (0 m deviation), confirming physical correctness, affirming the correctness of the gravitational force calculations, and demonstrating the numerical stability of the system under large-scale, long-duration integration.

### 5.2.2 RKF45 Integration Performance

The RKF45 (Runge-Kutta-Fehlberg) integration method was tested in a 10-day simulation using a 2.4-hour timestep with an orbital distance of 0.1% showing that spatial dynamics were preserved. The energy error remained at 0.4%, confirming the effectiveness of the adaptive step control and integration accuracy. This shows a correct implementation of all six RKF stages with proper intermediate derivative evaluations and weighting, ensuring high accuracy without sacrificing performance.

### 5.2.3 GPU Performance and Computational Throughput

To test scalability and GPU efficiency, simulations were performed with varying numbers of bodies, and performance was measured for both Newtonian and Barnes-Hut approaches:

|  |  |  |  |
| --- | --- | --- | --- |
| Bodies | Newtonian Time (μs) | Barnes-Hut Time (μs) | Speedup Factor |
| 100 | 31.3 | 510.7 | 0.06 |
| 500 | 33.0 | 5627.0 | 0.005 |
| 1000 | 30.6 | 1655.3 | 0.018 |

The Newtonian method consistently outperformed the Barnes-Hut implementation across all test scales. This is counterintuitive given the theoretical advantage of Barnes-Hut’s O(n log n) complexity compared to Newtonian’s Cambria Math O(n^2). However, the practical performance depends heavily on implementation details, tree balancing efficiency, and memory access patterns.

### 5.2.4 Observed Issues in Barnes-Hut Implementation

Despite passing the stability tests, an important limitation was observed: bodies under the Barnes-Hut integrator remained stationary during integration steps. This suggests that force calculations are likely being computed correctly, but position updates may not be applied properly, or the tree-based approximation introduces a level of smoothing that impedes motion at small system sizes. Further profiling and refactoring of the octree traversal or update stage is needed to achieve both correctness and efficiency.

### 5.2.5 Summary of findings

Strengths:

The Newtonian implementation is both accurate and computationally efficient for small-to-medium systems. The RKF45 integrator delivers strong temporal stability, and the CUDA kernels execute reliably for gravitational force accumulation and integration steps.

Areas for Improvement:

The Barnes-Hut kernel needs further optimisation and validation to become effective at scale. Additional logging, force visualisation, or per-body tracing may help isolate the issue.

The physics engine as a standalone module demonstrates a readiness for AI integration and further scaling. However, further enhancements needed for the Barnes-Hut algorithm would be beneficial to support very large simulations in future iterations.

## 5.3 AI Agent Evaluation

The AI-driven autonomous agent was evaluated across several areas, including mission performance, decision-making consistency, training convergence, memory system behaviour, and anomaly response. The agent operated in a simulated environment over 450 mission steps, guided by a transformer-based reinforcement learning model with hierarchical memory and dynamic mission phase awareness.

### 5.3.1 Mission Success and Performance Metrics

The agent successfully executed the full 450-step mission without failure. At the final step, the agent’s proximity to the target remained within 16,755.93 km, indicating navigational stability, albeit with some distance margin. Notably, the final fuel level was recorded at 121.00%, indicating potential over-activation of the refuelling behaviour.

* Total mission reward: 221.60
* Average reward per step: 0.4924

The reward trajectory shows moderate efficiency, suggesting a policy that maintains mission stability but may benefit from further optimisation, especially around energy management.

### 5.3.2 Decision-Making and Behavioural Patterns

Throughout the mission, the agent demonstrated low-latency responses and phase-aware action planning.

Phase usage:

* TRANSIT: 84.9%
* CRITICAL: 7.8%
* LAUNCH: 7.3%

This shows that most operations occurred in mid-mission transit conditions, with the agent appropriately transitioning to CRITICAL during hazard detection.

Action selection pattern:

* Dominated by adjust\_trajectory during TRANSIT
* Switched to refuel during the CRITICAL phase
* Behaviour consistent with encoded reward shaping and phase priorities

The agent’s response time to phase changes was immediate, demonstrating the effectiveness of the phase context integration into both decision-making and memory filtering.

### 5.3.3 Anomaly Handling and Environmental Adaptation

Between steps **439–450**, the simulation introduced radiation and toxicity hazards. In response, the AI agent:

* Entered CRITICAL phase
* Activated refuel behaviour with high frequency
* Demonstrated correct detection of environmental changes via anomaly subsystem

However, a behaviour anomaly was detected where its continued refuelling actions were executed despite fuel levels exceeding 100%, reaching 121%. This suggests a need for threshold-based refuelling constraints or a post-condition check for action suppression when the optimal state is achieved.

### 5.3.4 Training Regime and Policy Convergence

Training logs and reward metrics reveal.

Strong policy convergence:

* Agent favoured adjust\_trajectory in appropriate contexts
* Minimal reward variance after step 300, indicating stable learning

Checkpoints saved at regular intervals (250, 300, 350, 400, 450)

This consistency is aligned with the expected behaviour of deep Q-learning agents under stable policy regimes.

However, errors during model saving were observed:

* "float() argument must be a string or a number, not 'NoneType'"
* 'MarsExplorationMission' object has no attribute 'epsilon'

These exceptions, while not critical, indicate an issue with model serialisation or checkpoint callback routines. This may affect reproducibility and requires further investigation.

### 5.3.5 Hierarchical Memory System Usage

The AI system is designed around a three-tiered memory architecture for supporting both real-time control and long-horizon learning. While the current mission implementation does not instantiate these components as fully modular memory agents, several core structures fulfil these roles functionally and are integral to the AI agent's temporal reasoning capabilities.

The state\_history buffer effectively implements short-term memory (STM), capturing a rolling window of the most recent sensor readings and spacecraft states. This buffer:

* Maintains a fixed-length temporal sequence used as input for the Transformer model.
* Automatically discards the oldest state once new data is appended.
* Supports immediate reactivity by contextualising current input within recent history.

This STM mechanism enables the agent to remain responsive to short-term dynamics.

The experience\_buffer and normal\_data\_buffer together form the operational intermediate memory(IM), which:

* Persistently stores state-action-reward sequences across mission phases.
* Provides training samples for policy updates through experience replay.
* Buffers anomaly-free mission data to support unsupervised anomaly detection.

Unlike STM, IM retains data across multiple steps and phases, allowing for pattern recognition and learning across broader mission contexts. This contributes directly to policy convergence and behavioural stability.

While STM and IM are implicit, the LTM is explicitly implemented in the AI architecture. It supports:

* Episodic encoding of critical mission experiences.
* Persistent retrieval of high-reward trajectories to influence future decisions.
* Sequence-level pattern recognition through memory-based Q-value adjustment.

The agent operated under a three-tier memory structure produces these results

|  |  |  |  |
| --- | --- | --- | --- |
| Memory Type | implementation | Role | Observed Behaviour |
| STM | State\_history | |  | | --- | |  |  |  | | --- | | Stores current sequence window for the transformer | | Supports rapid responsiveness to mission transitions and anomalies |
| IM | experience\_buffer, normal\_data\_buffer | Buffers experiences for training and hazard profiling | Enables phase-aware policy learning and anomaly trend detection |
| LTM | Episodic memory module | Retains high-impact events for long-term guidance | High retrieval frequency of adjust\_trajectory with ≥0.3 reward |

Where memory utilisation was 74%, with high correlation between top-retrieved episodes and successful outcomes. This affirms the memory system’s role in sustaining temporal coherence and improving mission strategy over time.

### 5.3.6 Anomaly Detection and Hazard Response

In the late-stage mission logs, the functioning of the anomaly detection system:

* Activated in steps 439–450 in response to environmental changes (e.g., rising radiation).
* Peak anomaly prediction confidence reached 0.21, prompting refuel and emergency responses.

Despite successful activation, continued refuel actions post 100% fuel suggest the system lacked a suppression mechanism or post-condition awareness, which should be addressed in future training iterations.

### 5.3.7 Summary of AI Agent Performance

|  |  |
| --- | --- |
| Metric | Result |
| Mission Steps Completed | 450 |
| Final Fuel Level | 121.0% (overfilled ) |
| Final Distance to Target | 16,755.93 km |
| Avg. Reward per Step | 0.4924 |
| Policy Convergence | Strong (consistent Q-values and optimal action preference) |
| Anomaly Response | Activated successfully; context-aware |
| Model Stability | High, though minor checkpointing errors present |
| Memory Usage | 74% memory effectiveness |

The AI agent demonstrates competent autonomous control, effective reinforcement learning adaptation, and promising generalisation to new mission conditions. The architecture’s integration of temporal reasoning, hierarchical memory, and environmental awareness provides a solid foundation for scalable and adaptive AI systems.

## 5.4 Spacecraft Firmware Evaluation

The spacecraft firmware was evaluated through the Google Test Suite that focused on verifying the reliability, fault detection, memory safety, and synchronisation integrity of its core subsystems. It serves as a critical intermediary between the AI agent and the physics environment control layer, managing commands, telemetry, and memory transfers across multiple concurrent processes.

### 5.4.1 Test Overview

|  |  |
| --- | --- |
| Category | Status |
| Total Tests | 7 |
| Passed | 7 |
| Failed | 0 |
| Skipped | 0 |

All tests executed successfully, providing a good baseline level of confidence in the firmware’s stability and correctness.

### 5.4.2 Command Reliability

Basic operations:

The firmware accurately handles basic control commands like adjust\_trajectory or emergency\_protocol. All routine operations passed with 0 ms latency, indicating that the command parser and dispatcher are functionally correct.

Thread Safety:

The CommandReliabilityThreadSafety test validated concurrent access to the command queue and memory systems under multithreaded conditions. Proper std::mutex and atomic protections ensure safe and deterministic behaviour even when multiple threads interact with the firmware simultaneously.

Boundary Conditions:

The system demonstrates resiliencies under edge-case scenarios, such as invalid command values or memory overflows. No crashes or undefined behaviours were observed.

### 5.4.3 Fault Detection and Recovery

The FaultDetectionAndRecovery test confirmed the firmware’s ability to detect and react to operational anomalies. Internal mechanisms such as hasFault() and triggerEmergencyProtocol() were invoked in test scenarios simulating environmental faults and memory access violations.

* Implementation insight: Fault detection is integrated into the IntermediateMemory class via atomic flags (faultDetected, errorCount) and checkpoint restoration techniques.
* Emergency Mode Activation: If faults persist beyond acceptable thresholds, the system enters a fail-safe EMERGENCY state, halting thrust and preserving core telemetry.

### Telemetry Buffering and Synchronisation

Telemetry Accuracy:

The TelemetryBufferingAndSync test confirmed that sensor and system telemetry data were reliably transferred between memory modules (STM → IM → LTM). No data corruption or loss occurred during transfers.

Synchronisation Accuracy: The SynchronizationAccuracy test launched 5 concurrent threads writing to distinct memory regions. Thread locking via std::mutex prevented race conditions, and the final memory contents were validated against expected outcomes, confirming byte-for-byte integrity.

Thread Coordination: The test code utilised well-structured concurrency primitives (mutexes, atomic flags) to orchestrate thread-safe memory reads/writes and avoid deadlocks.

### 5.4.5 Evaluation Summary

|  |  |
| --- | --- |
| Criterion | Outcome |
| Command Execution | Reliable |
| Thread Safety | Verified |
| Fault Detection | Functional |
| Recovery Mechanisms | Basic |
| Telemetry Buffering | Accurate |
| Memory Synchronisation | Correct |

Strengths:

* All seven unit and system-level tests passed successfully.
* Fault detection is robust, and memory operations are thread safe.
* Memory subsystems implement checkpointing and verification, ensuring resilience.
* Modular design (via IntermediateMemory and SafetyManager) promotes maintainability.

Recommendations for Improvement:

* Implement stress testing with higher thread counts and randomised access patterns to validate under load.
* Expand fault injection scenarios to test recovery mechanisms, not just detection.
* Include integration-level testing with simulated AI and space environment modules to test telemetry end-to-end.

The firmware exhibits high reliability and resilience under simulated mission conditions. It acts as a robust middleware layer between the AI control logic and the underlying space environment, handling asynchronous communication, memory management, and fault containment. These results confirm the firmware’s readiness for integration into the full autonomous control system.

## 5.5 Integrated System Evaluation

Following the independent validation of the subsystem, they were integrated into one unified system via a ZeroMQ-based interposes communication (IPC) protocol. Two rounds of testing were done to assess the performance, communication reliability, and decision-making consistency of the entire system operating in real time.

### 5.5.1 Evaluation Metrics and Results

|  |  |  |  |
| --- | --- | --- | --- |
| Evaluation Category | Test 1 Result | Test 2 Result | Observation |
| Real-Time Responsiveness | Avg: 0.52 ms,  Max: 11.0 ms | Avg: 0.56 ms,  Max: 9.0 ms | Excellent response latency across both sessions |
| IPC Interoperability | 100% (50/50 messages) | 100% (50/50 messages) | Flawless bidirectional messaging via ZeroMQ |
| Autonomous Decision Alignment | 4/38 aligned (10.5%) | 13/34 aligned (38.2%) | Inconsistent alignment between AI actions and mission goals |

### 5.5.2 Real-Time Responsiveness

The integrated system demonstrates consistent real-time performance across both test sessions. The average response times remained below 1 ms, with maximum latency values well within acceptable operational limits (≤ 11 ms). This suggests that the CUDA-accelerated physics engine, spacecraft firmware, and AI inference pipeline are well-optimised for concurrent operation.

### 5.5.3 IPC Interoperability

ZeroMQ-based communication between all the subsystems achieved 100% message transmission and receipt in both test cases. All 50 requests in each test were successfully processed and acknowledged without delays or errors. This confirms the reliability of the IPC middleware and validates the robustness of the spacecraft firmware’s messaging handlers and JSON-based parsing infrastructure.

### 5.5.4 Autonomous Decision Alignment

Validation of the autonomous decision alignment is defined as the percentage of AI actions that matched mission objectives or external directives. This metric is essential in validating whether the AI not only functions internally but also interact meaningfully and correctly with the external simulation context. The results show that the decision alignment remains a major bottleneck:

* The first test showed poor alignment (10.5%), indicating that the AI either misunderstood command intent, overreacted to sensor inputs, or misinterpreted mission context.
* The second test showed measurable improvement (38.2%) following updates to anomaly detection thresholds and command preprocessing logic. However, this is still below expected accuracy for mission-critical scenarios.

These finding suggests that the misalignment is not attributed solely to deficiencies in the AI's internal policy or training regime. Instead, several broader system-level factors may have contributed:

* Outdate telemetry data where the AI operates on outdated or lagged data, resulting in actions that are well-intentioned but poorly timed for the current environment state.
* Reward attribution inconsistencies, where the AI may reinforce behaviours that are technically valid but do not align with the simulation’s internal notion of success or failure.
* verbal mismatches between the AI’s interpretation of state variables and the simulation’s intended meaning or thresholds.

These factors indicate a form of cognitive dissonance between the AI and the environment the AI is learning and adapting within its own internal frame of reference, but this frame may be misaligned with the simulation’s behavioural expectations. This underscores the logic, time-step synchronisation, and telemetry audit tooling during the integration phase.

Moving forward, there needs to be further improvements in telemetry structure standardisation, action-parameter validation, and real-time state coherence to enhance decision fidelity. Enhanced logging and joint protocol conformance testing between AI and simulation subsystems will be essential to close the alignment gap.

## 5.6 Summary of Findings

The research demonstrated the viability of integrating a Transformer-based AI agent, a CUDA-accelerated physics engine, and a modular spacecraft firmware into a cohesive autonomous space control system.

* Environmental: The Newtonian gravity simulation achieved excellent accuracy and stability with RKF45 integration. The Barnes-Hut implementation, though theoretically efficient, underperformed due to implementation limitations or insufficient system size.
* AI: It successfully completed mission sequences and showed strong phase-specific behaviour. It adapted to anomalies and made decisions based on learnt policies, although some inefficiencies were observed. Logging and memory systems functioned effectively, despite occasional checkpointing errors.
* Firmware: All firmware tests passed, confirming reliable command handling, fault detection, and memory synchronisation with thread safety.
* Integrated System: The system maintained real-time responsiveness and perfect IPC communication. However, autonomous decision alignment remained low in some scenarios, suggesting a possible mismatch between simulation outputs and AI expectations.

Overall, the system is functionally sound and modular, though improvements in decision consistency and inter-component data interpretation are needed for operational readiness.

# 6. Conclusion

This research presents the design, implementation, and evaluation of a modular autonomous spacecraft control framework integrating three core elements. The growing need for autonomy in deep-space missions, where operational risk, communication latency, and resource constraints limit the feasibility of continuous human supervision, motivated the research. This system aims to enable real-time, intelligent decision-making onboard the spacecraft itself.

The AI agent, powered by a transformer-based model, demonstrated strong contextual reasoning capabilities across mission phases and successfully adapted to dynamic conditions, including anomalies and fuel constraints. Its hierarchical memory system allowed for short-term responsiveness and intermediate pattern learning, while reinforcement learning ensured experience-driven policy development.

The physics engine delivered high-fidelity simulations of gravitational systems using adaptive Runge-Kutta integration and GPU acceleration. Although the Barnes-Hut implementation underperformed in smaller scenarios, the Newtonian solver offered excellent accuracy and performance. Meanwhile, the spacecraft firmware showed robust command execution, fault tolerance, and safe memory handling across all test cases.

While the system achieved high responsiveness and reliable interposes communication, the full integration revealed challenges in autonomous decision alignment, suggesting mismatches between AI expectations and environmental state representations. This highlights an important area for future refinement.

In the end, this work establishes a functional prototype for an AI-driven autonomous control system capable of interfacing with a simulation environment and acting independently of ground control. Laying a foundation for future enhancements in mission planning, anomaly response, and intelligent navigation in deep space exploration.

# References

McBrayer, K.T. *et al.* (2021) *Communication delays, disruptions, and blackouts for crewed Mars missions*, *NASA Langley Research Center*. <https://ntrs.nasa.gov/api/citations/20220013418/downloads/ASCEND-Communication%20Delays%2C%20Disruptions%2C%20and%20Blackouts%20for%20Crewed%20Mars%20Missions.pdf>.

Sutton, R.S. and Barto, A.G. (2014) *Reinforcement Learning: An Introduction*. Second edition, in progress. The MIT Press. <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf>.

Even-Dar, E., Kakade, S.M. and Mansour, Y. (2004) *Experts in a Markov decision process*. <https://proceedings.neurips.cc/paper_files/paper/2004/hash/421b3ac5c24ee992edd6087611c60dbb-Abstract.html>.

Doshi-Velez, F. and Cambridge University (2009) *The infinite partially observable Markov decision process*, *Cambridge University*. journal-article. <https://papers.nips.cc/paper_files/paper/2009/file/ebd9629fc3ae5e9f6611e2ee05a31cef-Paper.pdf>.

Aarseth, S.J. (2003) *Gravitational N-Body simulations*. https://doi.org/10.1017/cbo9780511535246.

*Matthijs Spaan: Partially observable Markov decision processes* (2012). <https://www.st.ewi.tudelft.nl/mtjspaan/publications/b2hd-Spaan12pomdp.html>.

Yin, H. et al. (2020) ‘ns3-ai: Fostering Artificial Intelligence Algorithms for Networking Research’, in *Proceedings of the 2020 Workshop on ns-3*. [Online]. 2020 New York, NY, USA: ACM. pp. 57–64.

McNaughton, M. *et al.* (2005) 'Broker: an interprocess communication solution for multi-robot systems,' *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1458–1463. https://doi.org/10.1109/iros.2005.1545040.

Córdoba, J.V.A.I. and Alfonso, M.J.F.I. (2022) 'Resolution of initial value problems of ordinary differential equations systems,' *Mathematics*, 10(4), p. 593. <https://doi.org/10.3390/math10040593>.

Brandt, A. (2022) *On Distributed Gravitational N-Body Simulations*. [Online]

Huan, Z. and Shijin, W., (2023) Applications and Challenges of Artificial Intelligence in Aerospace Engineering. [online] Available at: <https://ieeexplore.ieee.org/abstract/document/10206205/>.

Russo, A. and Lax, G. (2022) 'Using Artificial Intelligence for space Challenges: a survey,' *Applied Sciences*, 12(10), p. 5106. <https://doi.org/10.3390/app12105106>.

Maxion, R.A. (2025) *Wind turbine fault detection through AutoEncoder-Based neural networks*. <https://ieeexplore.ieee.org/document/10935128>.

Ayankoso, S. *et al.* (2024) 'Artificial-Intelligence-Based Condition monitoring of industrial collaborative robots: detecting anomalies and adapting to trajectory changes,' *Machines*, 12(9), p. 630. <https://doi.org/10.3390/machines12090630>.

*Garaffa, L.C. et al. (2021) 'Reinforcement Learning for Mobile Robotics Exploration: A survey,' IEEE Transactions on Neural Networks and Learning Systems, 34(8), pp. 3796–3810. https://doi.org/10.1109/tnnls.2021.3124466.*

Asiain, E., Clempner, J.B. and Poznyak, A.S. (2018) 'Controller exploitation-exploration reinforcement learning architecture for computing near-optimal policies,' *Soft Computing*, 23(11), pp. 3591–3604. https://doi.org/10.1007/s00500-018-3225-7.

Ge, L. *et al.* (2024) 'Deep reinforcement learning navigation via decision transformer in autonomous driving,' *Frontiers in Neurorobotics*, 18. <https://doi.org/10.3389/fnbot.2024.1338189>.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long Short-Term memory,' *Neural Computation*, 9(8), pp. 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

**Charles, E., Richard, L. and Ravi, Gondhalekar. "Autonomous six-degree-of-freedom spacecraft docking with rotating targets via reinforcement learning, (2021) Autonomous six-degree-of-freedom spacecraft docking with rotating targets via reinforcement learning. [online] Available at:** [**https://arc.aiaa.org/doi/abs/10.2514/1.I010914**](https://arc.aiaa.org/doi/abs/10.2514/1.I010914)**.**

Daniel, M. and Jakub, S., (2019) Research of an entity-component-system architectural pattern designed with using of data-oriented design technique. [online] Available at: <https://ph.pollub.pl/index.php/jcsi/article/view/1331>

Greaves, J. A. & Scheeres, D. J. (2023) Autonomous Optical-Only Spacecraft-to-Spacecraft Absolute Tracking and Maneuver Classification in Cislunar Space. *Journal of guidance, control, and dynamics*. [Online] 46 (11), 2092–2109.

Wikipedia contributors, (2025) *Runge–Kutta methods*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta\_methods [Accessed 02 Apr. 2025].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., (2017) Attention is All you Need. *arXiv (Cornell University)*, [online] 30, pp.5998–6008. Available at: <https://arxiv.org/pdf/1706.03762v5>.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A. and Mordatch, I., (2021) Decision Transformer: Reinforcement learning via sequence modeling. *Neural Information Processing Systems*, [online] 34. Available at: <https://dblp.uni-trier.de/db/journals/corr/corr2106.html#abs-2106-01345>.

Janner, M., Li, Q. and Levine, S., (2021) Offline reinforcement learning as one big sequence modeling problem. *Neural Information Processing Systems*, [online] 34. Available at: <https://openreview.net/forum?id=wgeK563QgSw>.

Baddeley, A., (1992) Working memory. *Science*, [online] 2555044, pp.556–559. Available at: <https://doi.org/10.1126/science.1736359>.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S.G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A.P., Hermann, K.M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K. and Hassabis, D., (2016) Hybrid computing using a neural network with dynamic external memory. *Nature*, [online] 5387626, pp.471–476. Available at: <https://doi.org/10.1038/nature20101>.

Ritter, S., Wang, J.X., Kurth-Nelson, Z., Jayakumar, S.M., Blundell, C., Pascanu, R. and Botvinick, M., (2018) *Been There, Done That: Meta-Learning with Episodic Recall*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1805.09692>.

Sakurada, M. and Yairi, T. (2014) 'Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction,' *dl.acm*, pp. 4–11. <https://doi.org/10.1145/2689746.2689747>.

Bird, S., Klein, E. and Loper, E. (2009) *Natural Language Processing with Python*.

Truszkowski, W.F. *et al.* (2006) 'Autonomous and autonomic systems: a paradigm for future space exploration missions,' *IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews)*, 36(3), pp. 279–291. <https://doi.org/10.1109/tsmcc.2006.871600>.

Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castaño, R., Davies, A., Mandl, D., *et al.* (2005) 'Lessons learned from Autonomous Sciencecraft experiment,' *ACM Conferences*, pp. 11–18. <https://doi.org/10.1145/1082473.1082798>.