RMIT UNIVERSITY

Deep Q-Learning for the Management of Distributed Satellite Systems

RMIT School of Engineering | Aerospace Faculty

Research Report

Authors

Jay Dickson | s3719855

Jeremy Bloor | s3787343

Supervisors

Dr. Andoh Afful

Professor Jennifer Palmer

Executive Summary

The focus of this project is to effectively manage Distributed Satellite Systems (DSS) in Low Earth Orbit (LEO) using Artificial Intelligence (AI) algorithms such as Deep Q-Learning Networks (DQN) to achieve a mission. The aim of this project is to access the feasibility of using DQN for autonomous management of DSS in a simulated environment. Therefore, the objectives of the project are to create a DQN agent that can independently follow a leader satellite and avoid hazards in a two-dimensional environment.

Python was chosen as the programming language to code and train the model as it has powerful libraries tailored for AI creation. A detailed literature review and methodology were carried out to ensure the project took an informed approach to solving the problem.

However, several challenges and roadblocks were experienced during the completion of the project such as long training times and the need for large computational resources. This is due to the limitations introduced as the iterative coding approach was impacted by long training times and complication with environmental modelling. As a result, the model demonstrated a random reward distribution throughout training, implying no model improvement occurred.

Following the unsuccessful completion of this project, several recommendations can be made to aid in further research. These are to understand the Markov Decision Processes deeper to better define the environment and include additional computational recourses in future projects to speed up training times.

Table of Contents

Table of Figures

Table of Tables

Table of Equations

1. Introduction

Distributed Satellite Systems (DSS) are low Earth orbit (LEO) missions that consist of two or more satellites working together to achieve a given mission that would otherwise be infeasible for a traditional monolithic satellite. However, as LEO becomes increasingly congested with debris from past space missions and increasing numbers of smaller satellites, new methods of managing these interconnected satellite systems must be devised [1]. One method to address this is with improved autonomy to manage each satellite and groups of satellites together to better achieve a mission [2] [3]. An innovative technique commonly used for control problems in robotics comes from Artificial Intelligence (AI) known as Deep Reinforcement Learning (DRL) such as Deep Q-Learning Networks (DQN) [4]. This combines the benefit of Deep Learning (DL) and Reinforcement Learning (RL) to achieve complex outcomes to complex inputs. By applying DQN to coordinating multiple satellites in real time, a large increase in capability can be achieved. The aim of this project is to access the feasibility of using DQN for autonomous management of DSS. To achieve this, the objectives of the project are to create a DQN agent that can act independently to follow a leader satellite and avoid obstacles. Due to the complexity, time constraints and computation power required to implement a DQN, no breakthrough results were achieved but with more time and resources a solution could likely be attained.

2. Background and Literature Review

2.1 Distributed Satellite Systems

The space environment has begun to be transformed by the deployment of DSS which are replacing older, static monolithic satellites due to the former's reduced size and weight alongside capabilities for redundancy and improved coverage. Many government and military projects are utilising constellations of ten or fewer satellites for missions rather than one large satellite. This is because larger numbers of small satellites can cover a greater area and are lighter and thus cheaper to launch into orbit. Additionally, if one satellite fails the mission can still be completed. These smaller satellites are often referred to as 'CubeSats' as they are made up of 10×10×10-cm cubes, each of which is dubbed a U as a measure of size. For example, a '12U' CubeSat has a volume of 12,000cm^3. Traditional monolithic systems are made up of several larger satellites such as GPS by the US government, Galileo by ESA and GLONASS for Russia. However, corporations aim to massively increase the number of satellites in orbit with SpaceX being the forerunner, currently orbiting 1,700 satellites in orbit with a planned constellation of 12,000 satellites. Amazon is also planning a constellation in the thousands and China's SatNet, Guowang is planning a 13,000-satellite constellation [1] [5] [6].

DSS are categorised into four main categories: constellation, train, fractionated and federated [7]. Constellation or swarm systems tend to be large, loosely coordinated formations of satellites. An example of a constellation is Starlink which is operated by SpaceX. They manage satellites with coverage of the Earth's surface used to provide internet service globally. Train or leader/follower arrangements normally are systems of five or fewer satellites where one satellite leads the other satellites in a particular orbit. Currently such strategies are used for Earth observation or to test reliance upon autonomous navigation and coordination systems such as in the case of NASA's Starling constellation [8]. Fractionated systems are comprised of groups of satellites, each of which performs a different task to achieve an overall mission. For example, one CubeSat generates power and provides it to several other 'observation' CubeSats that have cameras onboard and send data to a 'communications' CubeSat that has an antenna array which relays the data collected. Federated systems are groups of satellites which form in space locally to aid each other in achieving their goal. A method of organising federated systems is via a 'market' amongst them used to place 'wagers' to determine which can do what at different time intervals.

2.2 Resident Space Objects

Resident Space Objects (RSOs) are objects in the space environment. They include operational satellites, derelict satellites, and natural or artificial debris from past space missions and asteroids. As the number of operational satellites grows, so too does the risk of an accident. SpaceX filed a report in June 2023 with the United States Federal Communications Commission (FCC) claiming that 25,000 manoeuvrers were conducted across their Starlink constellation to avoid a collision [1]. The risk of collision exponentially grows as across time as more satellites are added LEO.

If RSOs of sufficient size collide, a cascade of further collisions could occur. This effect is known as Kessler Syndrome. The effect of such a cascade event could limit Earth operations for hundreds of years. The Kessler syndrome is named after NASA scientist Donald Kessler who predicted a possible scenario in LEO whereby the amount of debris accumulates

quicker than atmospheric drag can deorbit the objects [5] [9]. RSO debris range in size and number from 36,500 objects 10 cm wide to 130 million pieces 1 cm or smaller [10]. Due to the speeds that these objects circle the Earth, the potential for damage is high for satellites of any size. Clean-up efforts are limited in scope and scale as the objects left in space are owned by the country that left them there limiting cross nation efforts.

2.3 Coordination and Collision

As large numbers of satellites are deployed, they must also be controlled and managed to avoid hazards and to reposition to better achieve their given mission. Currently human operators comprise the main check if a satellite system will collide with RSOs based off US government data such as SpaceTrack. A possible future solution is system autonomy used to locally track and coordinate avoidance of debris via onboard sensors before relaying the orbit change information to a ground station. This autonomy would allow for quicker response times from multiple satellites that are part of a constellation and relieve the coordination capacity needed on the ground.

Partial low-level autonomy is included in most space systems due to bandwidth limitations and communication delays [11] [12]. Cramer, et al., in [8] outline when it is prudent to apply autonomous systems. They clarify that response time; performance improvement and risk awareness are significant considerations when choosing to implement autonomy. Araguz, et al., in [7] outline the key issues that autonomy can solve. They list mission robustness and tolerance to failures, improved data return, reduced visibility, and communication delays. All are significant considerations in the deployment of DSS formations and as such the application of robust autonomy is of paramount importance if they are to succeed at scale.

2.4 Deep-Q Network

To achieve the task of locally computing trajectories and repositioning, AI could be used. AI systems have progressed at a rapid rate in recent years, allowing for algorithms that can handle complex environments and produce complex actions as a response. Satellite formations are comprised of multiple individual satellites so the problem can be considered multi-agent [13] [14].

In the realm of satellite control and AI, there is a growing interest in leveraging AI techniques, notably RL, to facilitate the coordination of multiple satellites. RL is recognized for its capacity to train agents through a reward-based system, making it particularly relevant for complex control scenarios where satellites must interact with their environment [4]. One prominent approach within RL is Q-learning, which employs a lookup table and the Bellman equation [15] to make decisions based on the input states [16] [17]. The use of the Bellman equation allows for an estimate of the best action given a set of environmental parameters [18]. This process is expanded with DL, which replaces the table with a neural network expanding the number of state variables it can handle [4].

Within the training process, the concept of exploration vs exploitation plays a pivotal role, this is represented through the hyperparameter “epsilon” [2]. Higher epsilon values encourage agents to explore their environments by taking more random actions, while lower values prompt agents to exploit known pathways [2]. As epsilon decreases the training continues and so the agent explores less. The rate at which epsilon decreases is determined by a decaying exponential function that decreases each training cycle (episode).

To determine if training is completed, a ‘loss’ value is calculated. Loss is the measure of how close the predicted reward is to the actual reward gained [15] [19]. At the beginning of training the loss is high as the predicted rewards is far from the expected outcome but as training progresses, loss reduces. A common method for determining loss is the mean squared error function which uses the average squared difference between the predicted reward value and the actual reward value to qualitatively find the difference. Another commonly used method is the Huber Loss which is less influenced by outliers and so can be a better solution in some cases.

In a bid to enhance the stability of results achieved through DQNs, the concept of Double Deep Q-Networks (DDQNs) has been introduced. DDQNs utilize a secondary target network to assess Q-values before the primary Q-network's weights are updated, a strategy aimed at achieving more consistent and reliable results [4]. The target network is updated strategically at later stages in the training process, avoiding premature influence on the primary network.

2.5 Gap Analysis

The research completed in the field of LEO systems utilising autonomy is only part way in applying current and newly developed research from the field of AI. Many space missions conducted beyond Earth’s orbit utilise lower-level autonomy such as the Voyager

probe. However, these autonomous tasks primarily entail simple, pre-planned tasks. Deep space essentially empty and so mostly static, while the LEO environment is continually changing. Therefore, past autonomy methods, while useful must be built upon to tackle the LEO environment. For DSS to be successful and to scale effectively, autonomy must be created for each unit and between each unit to allow the system to function [20]. This will achieve a newer level autonomy. However, several conditions must be considered before better autonomy can be achieved: communication, collisions, and manoeuvring.

Herein focus primarily on the coordination to avoid debris and to better achieve the ongoing mission via manoeuvring. While communication between satellites and the ground stations is important, this is a sperate problem and so is not considered in this report. RSOs pose a large threat to DSS operations due to the ever-increasing number of new satellites, continuous levels of derelict satellites, debris and currently operating satellites. Manoeuvring multiple satellites in a timely manner while still achieving continuous operations for the required task would allow for a large leap forward in operational capability. For EO as an example, a phenomenon might be of interest in a different orbit trajectory, either inclination or azimuth, requiring the formation to reposition so it can collect data. As bushfires become a larger issue due to climate change, the tools available to monitor developing situations at a quicker rate will in turn allow quicker responses from emergency services. Developing autonomy to achieve this complex task for repositioning satellites while still in orbit will advance satellite capabilities dramatically while also saving time and cost by removing the need for additional satellites [3].

While AI has developed to a high capability for new and emerging problems, fewer applications have been made in the environment of space. Primarily RL has been used to simulate orbit manoeuvres but has not been used to control large systems of CubeSats [21]. This is due in part to the long training times required for RL and the large state spaces characteristic of DSS in LEO. DSS can be classes as a multi-agent problem due to the interaction of multiple ‘agents. Xie et al [4] demonstrates the use of DQN to solve this multi-agent problem by considering them in pair wise relationships. This reduces the computation time by considering the problem piece wise as well as using DL to make use of larger input states. Furthermore, Xie et al [4] demonstrates a target zone that the agent should aim for with a threat zone for the agent to avoid. By utilising the target zone, the agent can follow another agent at a distance and avoid obstacles that move into its threat zone. By applying DQN to DSS the issue of large inputs states and larger orbital systems can be addressed.

2.6 Objectives

The main objectives of this project are to:

1. Create a DQL Agent that can act independently to follow a leader node at a set distance and angle.
2. Have the Agent avoid obstacles and other hazards efficiently.
3. Show that a collection of Agents can act to optimise a formation of satellites while performing autonomous tasks.
4. Allow the formation to be managed through ground commands.

For DSS to be effective, the agents need to coordinate, avoid collisions, and complete tasks. The agent needs input data from the environment which it can then use to learn how it should act towards achieve its goal. Using this environmental data, the agent can be trained to avoid certain objects such as debris and to follow objects such as the lead satellite. By combining all these attributes, a satellite system should be able to act autonomously in a simulated environment to achieve a given mission.

3. Methodology

3.1. Overview

This section presents the methodology for implementing a DDQN algorithm tailored for the purpose of formation control and obstacle avoidance in multi-agent satellite systems. The following subsections detail the problem formulation, algorithm design, and simulation environment setup.

Fundamentally, the method is based around the theory of Markov Decision Processes. This academic area provides a framework to develop the most significant parameters needed for Deep Q-Learning such as the state space, action space and reward structure.

3.2. Problem Formulation

3.2.1. State Representation

The model's state representation consists of all the information the agent will have at a given step to make decisions with.

The information comes from the environment, which for any given episode will contain the agent which is dynamic as well as the target and a single obstacle,

these are stationary and randomly initialized randomly at the start of each episode. The state space for the model is defined as follows:

$$S = [x_t, y_t, v_{xt}, v_{yt}, x_o, y_o]^t,$$

Equation 1 - State Space of the Model

where x_t, y_t are the relative position of the agent with respect to the target, v_{xt}, v_{xt} which is the relative velocity of the agent with respect to the target and x_o, y_o which is the relative position of the agent with respect to the obstacle. This state is updated for each time step (t) in the episode.

3.2.2. Action Space

The action space represents the possible choices the model can make at a given time step. This is a discrete representation of the possible actions.

This is determined based upon the agent's discrete kinematic equation combined with the obstacles. The complete kinematic equation is defined below:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ v_{x,t+1} \\ v_{y,t+1} \\ x_{0,t+1} \\ y_{0,t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & dt & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & dt & 0 & 0 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \\ x_{0,t} \\ y_{0,t} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}$$

Equation 2 - Kinematic Equation

Given this structure the action space is defined as:

$$a = [u_x, u_y] \in$$
$$[0,0], [-2,0], [-1,0], [1,0], [2,0], [0,-2], [0,-1], [0,1], [0,2],$$

Equation 3 - Action Space

where u_x, u_y is the formation control command.

3.2.3. Reward Structure

The reward function serves to track the "value" of a given state. The actual value is somewhat irrelevant with the significance being how it changes over time. The reward ensures that the agent gets feedback on its actions and can thus optimize its performance.

The model relies on a reshaped reward function. This is to say that it not only gets rewarded for the outcome of a given episode but is also rewarded continuously throughout the episode. This is done to ensure the reward is not sparse and that the model has an idea of how a given action-state pair performs. This is essential during the learning process.

The reward function for outcomes is defined below:

$$R(s) = \begin{cases} 2, & \text{if finish} \\ -2, & \text{if collision} \\ -2, & \text{if out of range} \\ 0, & \text{otherwise} \end{cases}$$

Equation 4 - Objective Reward Function

and the reward function for the sparse rewards:

$$F(s,a,s') = \begin{cases} -\gamma\rho(s') + \rho(s), & \text{if } \rho_0 > 2d_{\text{safe}} \\ -\gamma\rho(s') + \rho(s) + \gamma\rho_0(s'), & \text{if } \rho_0 > 2d_{\text{safe}} \\ -\rho_0(s) + 2(1-\gamma)d_{\text{safe}}, & \text{if } \rho_0 \le 2d_{\text{safe}} \end{cases}$$

Equation 5 - Sparse Reward Function

where, s is the state at step t, a is the action at step t and s' is the state at step $t+1$. The coefficients γ and d_{safe} describe the value placed on future rewards and the safe distance to the target respectively. Finally, the functions $\rho(s)$, $\rho(s')$, $\rho_0(s)$ and $\rho_0(s')$ represent the agent's relative position to the target and its relative position to the obstacle. This is determined for the current step and the next step.

The two functions together allow for the calculation of the total reward for a given step.

3.3. Double Deep Q-Network (DDQN) Design

3.3.1 Network Architecture

The network structure is designed so that it can manage the complexities of the problem. The model has a total of five layers. One input, one output and 3 dense fully connected layers.

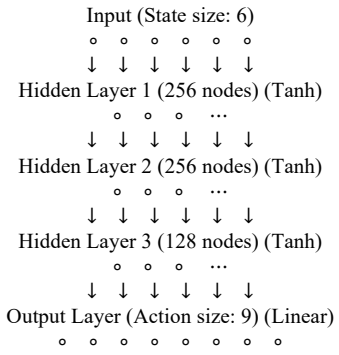

Figure 1: Neural network architecture of the Double Deep-Q Learning model

3.3.2 Learning Algorithm

3.3.2.1 Experience Replay

Experience replay is an integral mechanism in DQN that enhances both data efficiency and training stability. The algorithm stores tuples of past experiences, including state, action, reward, next state, and a 'done' flag, in a replay buffer. During the training process, mini-batches of these stored experiences are randomly sampled to update the Q-values. This approach serves two primary purposes: it increases

data efficiency by reusing each experience for multiple updates, and it enhances stability by breaking the temporal correlation in the sequence of observations meaning that the agent isn't trained too narrowly on data as further steps are based on previous steps.

3.3.2.2 Target Network Updates

To further stabilize the training process, DQN employs the concept of a target network. This network is essentially a clone of the primary Q-network but is updated less frequently. By using the target network to provide a more stable and consistent goal for Q-value updates, the overall stability of the training process is enhanced.

3.3.2.3 The Bellman Equation

DDQN improves upon DQN by employing a double Q-learning technique to reduce the overestimation bias associated with Q-value estimates. In DDQN, the action that maximizes the Q-value is selected by the primary network, but the evaluation of its Q-value is performed by the target network. The updated Q-value is calculated using the Bellman equation:

$$\text{Target} = R + \gamma \cdot Q_{\text{target}}\left(S', \text{argmax}_a Q_{\text{primary}}(S', a)\right)$$

Equation 6 - Bellman Equation

In this equation, the Target represents the goal for the Q-value to update to. R is the immediate reward, γ is the discount factor that modulates the influence of future rewards, and s' is the next state.

3.3.2.4 Optimization Techniques

The network parameters are updated using the Adam variant of gradient descent. Adam is favoured for its adaptive learning rate capabilities, which are particularly beneficial for navigating the high-dimensional parameter spaces encountered in DQN models.

3.3.3 Hyperparameters

HYPERPARAMETER	VALUE	DESCRIPTION
GAMMA	0.95	The discount factor used to balance immediate and future rewards. Values range from 0 to 1. A high value like 0.95 emphasizes the importance of long-term rewards.
EPSILON	1	Initial exploration rate. This determines the likelihood of taking a random action. A value of 1 means the agent will always explore initially.
EPSILON MIN	0.01	Minimum exploration rate. As learning progresses, epsilon decreases but won't go below this value. This ensures some level of exploration continues.
EPSILON DECAY RATE	10	Number of episodes to decay epsilon. This sets how often the exploration rate will be updated.
EPSILON DECAY	0.01	The value epsilon decays by. This is a small value to gradually decrease the exploration rate.
LEARNING RATE	0.01	The learning rate used in the Q-learning update. It determines the weight given to new updates.
NETWORK REPLACE	2000	Number of steps before replacing the target network. The target network is updated to match the weights of the main Q-network every N replace steps.
BATCH SIZE	32	The number of experience samples from the replay memory that are used for training the Q-network in each iteration.
MEMORY	100000 (max length)	A deque data structure with a maximum length of 100000. It is used for experience replay, storing tuples of (state, action, reward, next_state, done) for training.

3.4 Simulation Environment Setup

3.4.1 Satellite Models

The simulation is based on a simplified two-dimensional Cartesian coordinate system, focusing primarily on the kinematic aspects of satellite motion. This planar model allows for the simulation of a flat orbit around a central gravitational body, serving as a computationally efficient approximation for satellite behaviour.

3.4.2 Scenario Configuration

The model focuses on scenarios involving a single satellite agent, which is tasked with maintaining formation with a leader node while avoiding space debris. The leader node serves as a reference point, and the agent's objective is to maintain its relative position to this leader as seen in Figure 2 below.

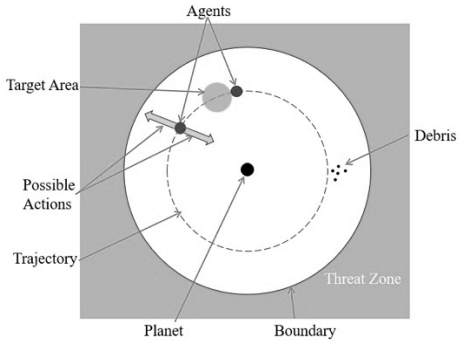

Figure 2: DSS configuration

In each scenario, there are two primary entities: the agent satellite and the leader node. Both satellites have predefined initial positions and velocities. The leader node follows a predefined trajectory, which the agent satellite aims to mirror while avoiding obstacles.

The primary focus is on maintaining a specific formation with the leader node, typically defined by a set distance and angle relative to the leader's position. The agent uses its control mechanisms or actions to adjust its position and velocity to achieve and maintain this formation.

Stationary space debris serves as the main type of obstacle in these scenarios. Given the two-dimensional nature of the model, the debris is represented as a circle within the orbital plane, which the agent must navigate around.

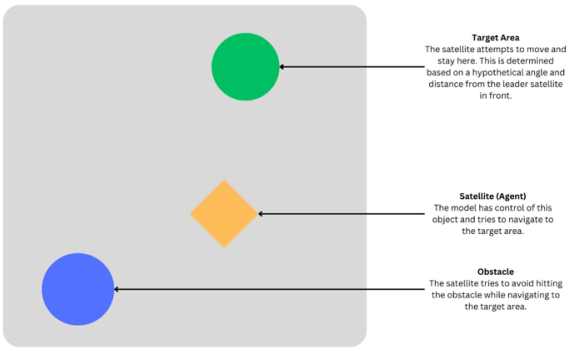

Figure 3: Training configuration

3.4.3 Evaluation Metrics

The primary metrics used for evaluating the performance of the Deep Q-Network (DQN) model in managing the satellite formation are loss and reward values. These metrics serve as indicators of how well the satellite agent is learning to achieve its objectives, which include maintaining a specific formation with the leader node and avoiding obstacles like space debris.

3.4.3.1 Loss Values

The loss values represent the difference between the Q-values predicted by the DQN and the actual Q-values calculated based on rewards received and future state estimates. A decrease in the loss value over time suggests that the model is learning to better approximate the optimal Q-function. Monitoring loss values is crucial for understanding how well the neural network is being trained to estimate future rewards.

3.4.3.2 Reward Values

The reward values indicate the immediate benefit the agent receives from taking specific actions in particular states. The rewards are designed to reflect real-world objectives, such as maintaining a specific distance and angle relative to the leader node and avoiding collision with obstacles. An increase in the cumulative reward over time implies that the agent is learning to take more optimal actions to meet its objectives.

3.5 Experimental Setup

To achieve the objectives, a range of software tools and cloud-based computing resources was utilized. Python was selected as the primary programming language because of its wide-ranging applications in both machine learning and scientific research. This language provided the foundational base for the computational tasks. Building upon this, TensorFlow was employed as the machine learning framework. TensorFlow was ideal for constructing and training the DQN model due to its highly optimized libraries tailored for machine learning tasks.

For the visualization aspect of the experiment, Pygame was chosen for rendering the training visuals. This allowed for the effectively display of the environment in which the satellites operated, providing valuable visual feedback. In addition, NumPy was used for several numerical computations and data manipulation tasks.

All simulations and training processes were executed on Paperspace, a cloud computing platform known for its high-performance capabilities. This service allowed the models to run through Jupyter Notebooks and gave access to GPUs and memory capacity that would not have been accessible otherwise.

4. Scope and limitations

The main constraints of the project were time and computational resources. Running the model on cloud-based infrastructure took approximately 30 hours, and the duration was even longer when using personal devices for testing, as the aim was to minimise cloud-computing costs. These time constraints hindered progression due to the inability to engage in iterative problem-solving effectively making the troubleshooting of implementation issues more challenging.

Additionally, due to these limitations, the scope of the project was narrowed to focus only on a 2D environment with a single agent. While this simplification enabled progress to occur, it also posed limitations on the applicability of the findings. Specifically, the reduced scope doesn't fully capture the complexities of orbital dynamics, which could impact the generalisability of the model to real-world, three-dimensional space scenarios.

5. Results

The project faced challenges in gaining momentum, primarily due to the realisation that many of the initial approaches were either unfeasible or too complex to implement within the constraints. As it stands, the project did not yield a viable model for satellite formation control using Deep Q-Learning. The statistical analysis underscored this outcome with the reward for the most recent model returning a P-value of approximately 1, this indicates that the data was randomly distributed. This suggests that the model did not learn any meaningful pattern or strategy for achieving its objectives, casting doubt on its current effectiveness and applicability.

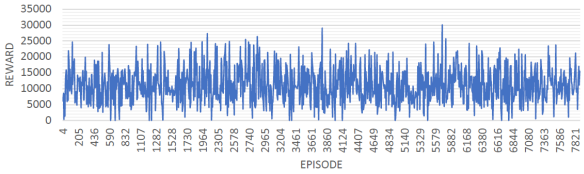

Figure 4: Reward progress for each episode

The reward graph represents the cumulative reward gained per training cycle or episode as seen in Figure 4. Each episode the reward gained is governed by the reward function outlined in reward function equation. This function maps the actions in the environment to rewards or penalisations and should trend upwards to show that the agent is improving at accumulating more reward with some variation. However here it is flat meaning the agent didn't learn anything meaningful.

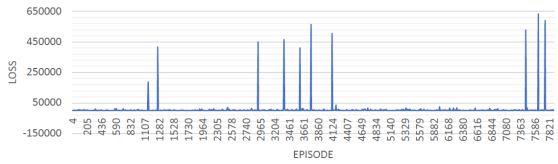

Figure 5: Loss for each episode

The loss function represents the model's accuracy. It is evaluated based on comparing the prediction the model makes with the actual result. This is calculated with the mean squared error function which uses the average squared difference between the predicted reward value and the actual reward value to determine the loss.

6. Discussion

This project did not determine whether DQL can serve as a viable method for autonomous satellite management. Overall, the approaches taken early in the project were not viable. As iteration is difficult to do regularly, evaluating how inefficiencies were being introduced was problematic. A balance was struck between changing a small enough set of variables to identify which one of them caused the problem, while also reducing the number of times the full model had to be run. This proved to be one of the most difficult parts of troubleshooting.

Since concluding this project, several errors have been identified in the implementation. One of which is that the action values were not correctly implemented. These values are meant to be passed into the kinematic equation to then determine the agent's movement but were instead represented as velocity pairs. The near perfect distribution of the reward values also implies that the model is not correctly acting on its model. Possibly a result of the action value mishap but it could also be a separate error that was never identified, as this issue seemed to have been the main prevailing problem, even during separate tests with a different model.

The results clearly show no improvement in the model's performance. All prior models also notably performed worse than the newest model. This implies that the most recent approach was improving the performance but had yet to be implemented fully.

The results would show an increase in the reward and a drop in the loss if the model was improving. This is not seen and so it can be concluded that the model did not learn well.

Overall, the results do not support the outlined hypothesis that DQL can serve as a viable method for distributed satellite management.

7. Recommendations

Moving forward, several actionable recommendations can be made that can significantly expedite progress in this research area.

Firstly, diving deeper into the fundamentals of Markov Decision Processes is highly recommended. This framework provides a systematic, mathematically rigorous way to define state space, action space, and reward function, focusing on this avenue will ensure the model is well defined.

Further, caution should be taken when contemplating an increase in environmental complexity. While this may seem like a logical step for improving the model, it could exponentially amplify the number of state variables, adding complexity that may become unmanageable. Any increase in complexity would also necessitate a thorough re-evaluation of the associated Markov Decision Process to adapt it to the new set of kinematic conditions.

Finally, investing in more powerful computational resources should be a priority, as it will accelerate both the development and validation processes. This is due to most of the work revolving around the need to develop and train neural networks. These networks are complex and time consuming to develop. Initially it was assumed that such recourses were freely available and ready to be used but these resources were limited for use by postgraduate students. The solution was to run the model on the cloud-computing platform Paperspace, this however involved a monetary cost.

8. Conclusion

In summary, despite encountering various challenges and ultimately not achieving success in the project, the knowledge gained, and suggestions put forward serve as a valuable starting point for future research. Most importantly concentrating on the theoretical aspects of Markov Decision Processes, which can lead to the development of robust and precisely defined models. Additionally, prioritizing computational resources and optimizing the model's exploration in complex environments is crucial. Although this area presents numerous difficulties, it also presents significant opportunities for making meaningful advancements. Through the insights highlighted, a worthwhile contribution has been made to the ongoing and future investigations in this field.

References

[1] T. Pultarova, “SpaceX Starlink satellites had to make 25,000 collision-avoidance maneuvers in just 6 months — and it will only get worse,” Space.com, 7 July 2023. [Online]. Available: https://www.space.com/starlink-satellite-conjunction-increase-threatens-space-sustainability. [Accessed 20 September 2023].

[2] K. Doshi, “Reinforcement Learning Explained Visually (Part 3): Model-free solutions, step-by-step,” Medium, 31 October 2020. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-explained-visually-part-3-model-free-solutions-step-by-step-c4bbb2b72dcf. [Accessed October 2023].

[3] E. Lagona, S. Hilton, A. Afful, A. Gardi and R. Sabatini, “Autonomous Trajectory Optimisation for Intelligent Satellite Systems and Space Traffic Management,” *Acta Astronautica,* pp. 1-1, 2022.

[4] N. Xie, Y. Hu and . L. Chen, “A Distributed Multi-Agent Formation Control Method Based on Deep Q Learning,” *Frontiers in Neurorobotics,* vol. 16, 2022.

[5] M. Wall, “Kessler Syndrome and the space debris problem,” Space.com, 15 July 2022. [Online]. Available: https://www.space.com/kessler-syndrome-space-debris. [Accessed 20 September 2023].

[6] A. Jones, “China is developing plans for a 13,000-satellite megaconstellation,” SpaceNews, 21 April 2021. [Online]. Available: https://spacenews.com/china-is-developing-plans-for-a-13000-satellite-communications-megaconstellation/. [Accessed 20 September 2023].

[7] C. Araguz, E. Bou-Balust and E. Alarcon, “Applying Autonomy to Distributed Satellite Systems: trends, challenges and future prospects,” *Technical University of Catalonia,* pp. 1-1, N-A.

[8] N. Cramer, D. Cellucci, C. Adams, A. Sweet, H. Mohammad and J. Frank, “Design and Testing of Autonomous Distributed Space Systems,” in *35th Annual Small Satellite Conference*, Moffett Feild, 2021.

[9] D. J. Kessler and B. G. Cour-Palais, “Collision frequency of artificial satellites: The creation of a debris belt,” *Journal of Geophysical Reserach Space Physics,* vol. 83, no. A6, pp. 2637-2646, 1978.

[10] European Space Agency, “Space debris by the numbers,” 12 September 2023. [Online]. Available: https://www.esa.int/Space_Safety/Space_Debris/Space_debris_by_the_numbers. [Accessed 20 September 2023].

[11] D. Atkinson, S. Chien and E. Mjolsness, “APPLICATIONS OF ARTIFICIAL INTELLIGENCE FOR SPACECRAFT AUTONOMY AND ENHANCED SCIENCE DATA RETURN,” in *AIAA Space 2000 Conference & Exposition*, Long Beach, CA, 2000.

[12] T. Rupp, S. D'Amico, O. Montenbruck and E. Gill, “AUTONOMOUS FORMATION FLYING AT DLR’S GERMAN SPACE OPERATIONS CENTER (GSOC),” *N-A,* pp. 1-1, N-A.

[13] S. D'Amico, Autonomous Formation Flying in Low Earth Orbit, Ridderprint BV, 2010.

[14] F. Bauer, K. Hartman, J. How, J. Bristow, D. Weidow and F. Busse, "Enabling Spacecraft Formation Flying through Spaceborne GPS and Enhanced Automation Technologies," in *1999 ION-GPS Conference*, Nashville,, 1999.

[15] K. Doshi, "Reinforcement Learning Made Simple (Part 2): Solution Approaches," Medium, 24 October 2020. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-made-simple-part-2-solution-approaches-7e37cbf2334e. [Accessed October 2023].

[16] TF-Agent Authors, "Introduction to RL and Deep Q Networks," TensorFlow, 26 September 2023. [Online]. Available: https://www.tensorflow.org/agents/tutorials/0_intro_rl. [Accessed October 2023].

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," in *NIPS Deep Learning Workshop 2013*, 2013.

[18] K. Doshi, "Reinforcement Learning Explained Visually (Part 4): Q Learning, step-by-step," Medium, 28 November 2020. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-explained-visually-part-4-q-learning-step-by-step-b65efb731d3e. [Accessed October 2023].

[19] K. Doshi, "Reinforcement Learning Explained Visually (Part 5): Deep Q Networks, step-by-step," Medium, 20 December 2020. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b. [Accessed October 2023].

[20] P. Oche, G. Ewa and N. Ibekwe, "Applications and Challenges of Artificial Intelligence in Space Missions," *IEEE Access,* pp. 1-1, 2020.

[21] Y. Sanz, J. de Lope and J. Antonio Martín H, "Applying Reinforcement Learning to Multi-robot Team Coordination," in *Third International Worshop HAIS*, Burgos, 2008.

Appendix 1- Contribution Table

Section	**Person(s) responsible and percentage**
Executive Summary	Jeremy Bloor 100%
Introduction	Jeremy Bloor 100%
Literature Review	Jeremy Bloor 100%
Methodology	Jay Dickson 100%
Results	Jay Dickson 100%
Discussion	Equal contribution
Conclusion	Jay Dickson 100%
Recommendations	Equal contribution