AUTONOMOUS SENSOR TASKING AND OBJECT DETECTION FOR SPACE SITUATIONAL AWARENESS USING MACHINE LEARNING

MICHAEL FAIRBROTHER

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN PHYSICS AND ASTRONOMY
YORK UNIVERSITY
TORONTO, ONTARIO

FEBRUARY 2024

© MICHAEL FAIRBROTHER 2024

Abstract

The space domain is becoming increasingly crowded due to the proliferation of man-made resident space objects. Our limited number of object-tracking resources is becoming overburdened by a growing catalog of space debris and active satellites. This problem necessitates the most effective use of available sensors and observatories for catalog maintenance through the optimization of sensor tasking schedules. Given the complexity of the tasking problem, autonomous scheduling is essential and creates a need for increasingly powerful computational optimization algorithms. This research explores one such optimization approach, deep reinforcement learning, as an effective solution for autonomous sensor tasking. In addition to scheduling, improved object detection algorithms are in constant demand in order to more accurately identify resident space objects in images produced by optical sensors. This research additionally explores region-based convolutional neural networks as a potential considerable improvement over traditional detection algorithms.

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Description
SSA	Space Situational Awareness
RSO	Resident Space Object
SSN	Space Surveillance Network
LEO	Low Earth Orbit
ML	Machine Learning
YOLO	You Only Look Once
CNN	Convolutional Neural Network
R-CNN	Region-Based Convolutional Neural Network
FOM	Figure of Merit
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
STK	Systems Tool Kit
NSGA-2	Non-Dominated Sorting Genetic Algorithm II
EOS	Earth-Observing Satellite
MDP	Markov Decision Process
SARSA	State–Action–Reward–State–Action
TD	Temporal Difference
DQN	Deep Q Network
A2C	Advantage Actor Critic
GAN	Generative Adversarial Network
PPO	Proximal Policy Optimization
GEO	Geostationary Equatorial Orbit
SGP4	Simplified General Perturbations 4
FOV	Field of View
FOR	Field of Regard
ACS	Ant Colony System

Acronym	Description
DQL	Distributed Q-Learning
TLE	Two-Line Element
SB3	Stable-Baselines3
CASSIOPE	Canadian Space Agency's Cascade SmallSat and Ionospheric Polar Explorer
FAI	Fast Auroral Imager
NEOSSat	Near Earth Object Surveillance Satellite
SNR	Signal-to-Noise Ratio
KL	Kullback-Leibler
RPN	Region Proposal Network
RoI	Region of Interest
FPN	Feature Pyramid Network
FPGA	Field-Programmable Gate Array
SVM	Support Vector Machine
IoU	Intersection Over Union
NMS	Non-Max Suppression
MEO	Medium Earth Orbit
SSO	Sun-Synchronous Orbit
GSO	Geosynchronous Orbit
HEO	Highly Elliptical Orbit
RAAN	Right Ascension of the Ascending Node
ECI	Earth-Centered Inertial

1 Introduction

Space Situational Awareness (SSA) is a vital area of scientific research whose importance only continues to grow as space becomes more densely populated with man-made satellites and debris. Over the last several decades there has been a significant proliferation of man-made "resident space objects" (RSOs) in orbit around Earth that have created a substantial crowding problem. These RSOs present ongoing potential collision threats to operational space assets [1, 3], challenging the Big Sky Theory. For context, RSOs are any objects found in orbit around Earth ranging from debris – both man-made and natural – to active satellites. Most of this space debris is very small, sometimes less than a millimeter in diameter [4, 5].

Unfortunately, any one of these pieces of debris, regardless of size, represents a serious ongoing collision threat to the many operational space assets that we collectively rely on every day for GPS navigation, internet, communications, and a host of other vital functions, including those related to national security. Given that the number of new satellites will continue to increase at an exponential rate, especially with the continued development of satellite constellations [2], it is imperative that we continue making concerted efforts to identify and regularly track as many of these objects as possible.

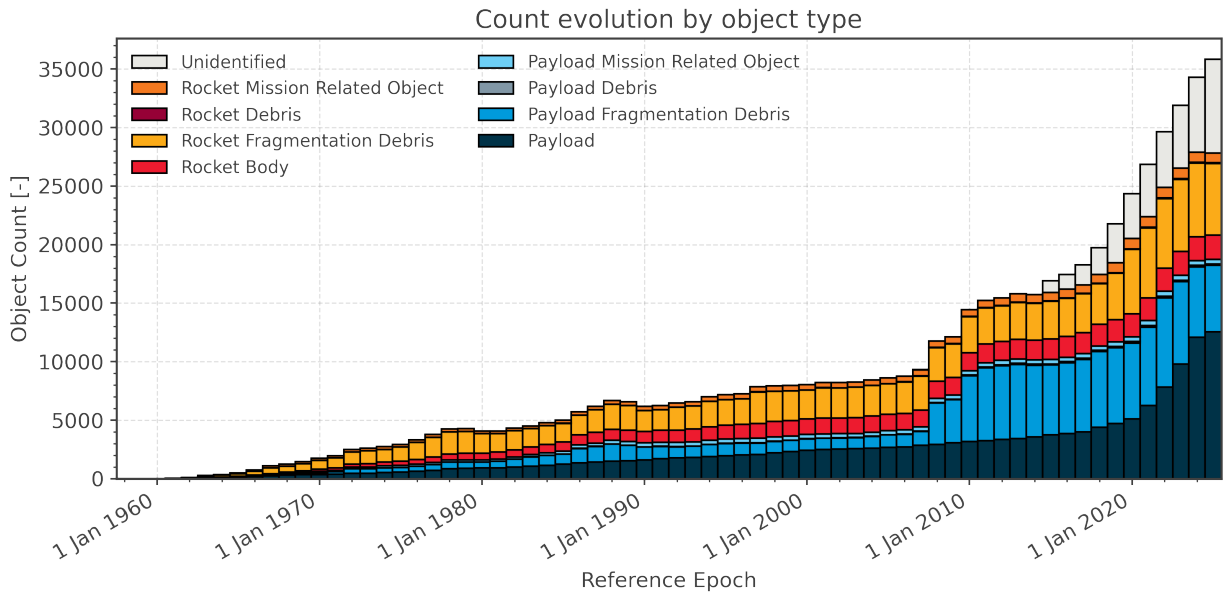


Figure 1: Graph showing the increase of space debris in orbit around Earth over time, broken down by type [5].

Apart from satellites and other similarly large objects, even pieces of debris smaller than 1 cm [4] can cause significant and irreparable damage to expensive and sensitive equipment when traveling at orbital speeds, and larger objects can easily render any satellite completely inoperable. A collision with a sufficiently large object would likely create clouds of additional debris, further exacerbating the crowding problem; something that has happened in the past to devastating effect [6]. In 2009 for instance, a decommissioned Russian satellite known as Cosmos 2251, with a mass of about 900 kilograms, collided with the 500-kilogram American commercial satellite known as Iridium 33. This created a cloud of thousands of additional pieces of debris. This collision represented a clear and direct challenge to the "Big Sky" paradigm that has been the dominant notion in the space exploration domain for decades when it comes to perceptions of collision risk. Efforts to track RSOs in order to prevent these types of events form the basis of the SSA initiative.

One of the main goals of the field of SSA is to encourage governments and private sector organizations that are active within the space domain to focus on increasing their awareness (that is, the tracking and prediction) of RSO motion to ensure the continued safe operation of important space assets. As a goal of SSA, this can include efforts aimed at increasing cooperation between governments and other organizations in sharing information regarding their space operations, as well

as efforts to identify new objects and track known objects. These efforts may involve researching, designing, and deploying new hardware, or researching and developing various software solutions.

The United States, for example, tracks around 27,000 pieces of space debris through its Space Surveillance Network (SSN). The SSN is a constellation of dedicated satellites for debris tracking. The United States cannot shoulder the burden of RSO cataloging and tracking alone, of course. Under Article IX of the UN Outer Space Treaty, states must ensure that their activities in space are conducted with some degree of SSA so as not to interfere with the activities of other state parties [7]. The UN Registration Convention also stipulates that state parties must keep a registry of objects that they send into space and that they must be willing to share tracking data related to those objects with other governments.

The field of SSA can be categorized into five fundamental pillars, according to the US Air Force [8]. The first of these is detection, tracking, and identification. For low Earth orbit (LEO) RSOs this is typically accomplished via radar, whereas objects in geostationary orbits are detected and tracked with optical sensors.

Characterization is the next pillar of SSA. This involves the establishment of an object's behavior such as attitude and orbit. This allows satellite operators to identify changes in an object's patterns and make decisions regarding its potential impact on other RSOs.

Beyond the commercial and civilian domain, satellites can be vital military assets for surveillance and communications. As hostile nations continued to develop their own space programs, they have also explored the possibility of deploying weapons in space as a means to harm military satellites. China and Russia have both developed anti-satellite missiles, for instance. Both nations have also already demonstrated the capability of these weapons. This pertains to the third pillar of SSA, which is tactical warning and attack assessment. SSA is a requirement for identifying hostile actions by other nations.

The fourth pillar of SSA is data integration and exploitation. This involves more effective data processing and reduction techniques in order to gain further insights. Lastly, spacecraft protection and resiliency make up the final key pillar. This involves research into features for satellites and other

spacecraft that could make them more durable, more maneuverable, or more aware by outfitting them with more sensors. In the case of satellite constellations, having some satellites take over tasks of another satellite when it becomes damaged also increases resiliency.

1.1 Space Situational Awareness

The SSA problem is a multi-faceted problem of considerable scale and complexity, and this complexity is only growing with every new piece of equipment launched into space. As the number of orbiting RSOs continues to increase, their numbers continue to rapidly outpace the number of dedicated observatories and space-based RSO tracking platforms that we have at our disposal. As a consequence of this, coordinating object tracking efforts between our various relevant assets to maintain adequate SSA becomes a significant scheduling optimization challenge. Scheduling or tasking in SSA refers to the assignment of various tasks to ground and space-based assets. Besides object tracking, observatory assets also require time for data uplinking, downlinking, or maneuvering, in the case of space-based assets. Each of these tasks is subject to all of the associated real-world constraints that need to be considered as well, making the challenge that much more daunting. Some of these constraints might include data storage capabilities, availability of power, camera slew rates, and weather interfering with observations. An optimization problem of this level of complexity, exemplifying the "curse of dimensionality" [9], requires robust optimization algorithms that can create schedules in accordance with various constraints, while also ensuring the most optimal use of the limited resources available.

In addition to the problem of efficiency and optimization, the search for more powerful autonomous object detection and classification algorithms continues to expand. There are state-of-the-art computer vision machine learning (ML) algorithms such as You Only Look Once (YOLO) [10], or the family of convolutional neural networks (CNNs) known as region-based convolutional neural networks (R-CNNs) [11, 12, 13], that could have great potential in an SSA context but have seen limited use for SSA to date. These modern object detection and classification algorithms each have their limitations and advantages over one another – and over non-ML object detection and tracking algorithms – but ultimately, they could prove to be powerful solutions to the RSO detection and tracking problem for SSA. After all, their effectiveness has already been demonstrated in a range

of everyday image-processing tasks, such as autonomous driving. Object detection algorithms based on neural network architecture are very robust and generalizable. Once they are taught sufficiently well, they can detect and classify many different types of objects ranging in visual complexity with high accuracy; and they can do so completely autonomously.

1.2 Research Objectives

In establishing the scope of research for this thesis, challenge areas of SSA were identified where modern ML techniques could contribute significantly. In particular, certain aspects of SSA were chosen that fall within the purview of remote optical sensing. These aspects are object detection with optical sensors and optical sensor tasking. When it comes to object detection, there is a wide range of sophisticated object detection algorithms that make use of different neural network architectures. Many of these have yet to be widely used in real SSA scenarios. Two major examples are YOLO and R-CNNs. Both are advanced object detection frameworks that have been adopted in numerous real-world applications, such as surveillance and facial recognition. There are also other network architectures that have seen some consideration in SSA tasks like EfficientDet [33].

In the case of sensor tasking, any object-tracking and observation efforts depend on the use of a limited number of space and ground-based sensors. Therefore, RSO observations and tracking tasks must be scheduled, often between a number of stations, in order to make the best use of optical sensors and maximize the time spent observing particular figures of merit (FOM) while avoiding overlapping and redundant observations. Many constraints must be considered in real-life sensor tasking, and this requires powerful scheduling techniques with scalability in order to adequately incorporate these constraints into optimized solutions.

1.3 Orbital Mechanics

RSO motion is governed by Newton's law of gravitation as well as Kepler's laws. Masses in orbit around a central massive body like Earth are under the constant influence of gravitational force, described by Newton's formula for gravitational force.

$$F_g = \frac{GMm}{r^2} \tag{1}$$

The lower-case mass typically denotes the mass of the orbiting body around the more massive body. G is the gravitational constant, which acts as a scaling factor and ensures that the units are consistent in calculating the force of gravitational attraction between two bodies. This force of attraction is inversely proportional to the square of the distance between the two bodies.

In order for an orbiting body to maintain a stable orbit, the centripetal force of the object must counteract the force of gravity.

$$F_c = \frac{mv^2}{r} \tag{2}$$

The centripetal force is directly proportional to the square of the orbiting object's velocity, and inversely proportional to the distance to the center of the larger mass. When the two forces are balanced, the velocity required for RSOs to remain in a stable orbit would be given by the orbital velocity equation.

$$v = \sqrt{\frac{GM}{r}} \tag{3}$$

Kepler's laws further govern the motion of an RSO around Earth, as well as the shape of the RSO's orbit. A generalized version of Kepler's laws is as follows:

1. An object's orbit around a central body is an ellipse, with the central body's center of mass at one focal point of the ellipse.

2. An orbiting object sweeps out equal areas in equal time intervals along the orbital path.

3. The square of the orbital period of an object is proportional to the cube of the semi-major axis of the orbit.

$$T^2 \propto a^3 \tag{4}$$

What the first law tells us is that an orbiting body that orbits around a central body will have an elliptical shape due to the interaction of gravitational force with the initial velocity of the orbiting object. The degree to which the orbit is elliptical is known as the orbit eccentricity.

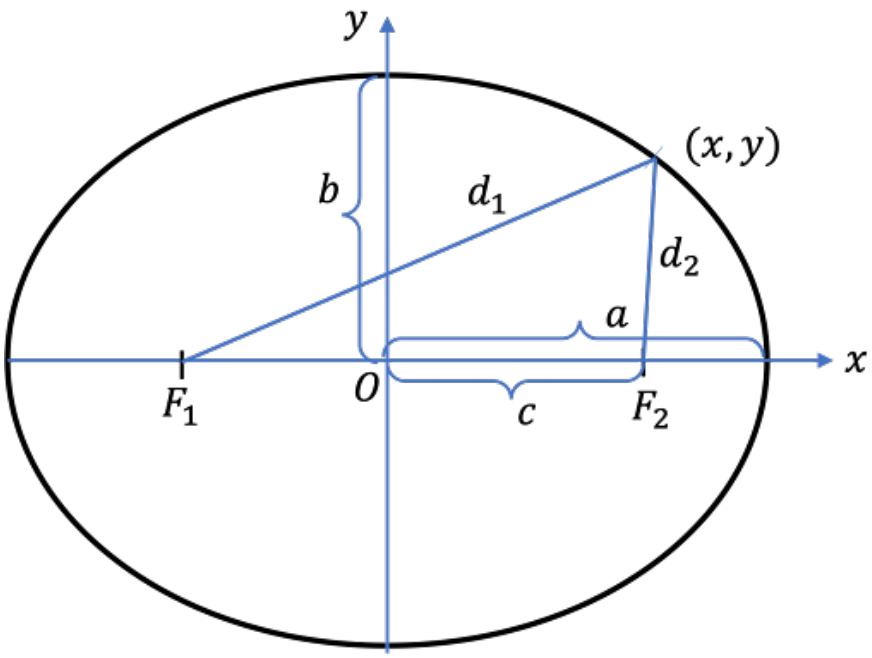


Figure 2: Diagram of elliptical orbit, displaying major and minor axes with geometric focal points included [50].

An ellipse has two axes, major and minor. The semi-major axis "a" is the distance from the center of the ellipse to the furthest point on the ellipse, whereas the semi-minor axis "b" is the distance to the closest point. Ellipses have two geometric focal points that sit along the major axis.

The second law states that an orbiting object sweeps out equal areas in equal times. This pertains to how the motion of the object changes as it approaches the periapsis. Closer to the periapsis, the object travels faster as the force of gravity is stronger. What the second law tells us is that the amount of area covered by the orbiting object is equal for the same orbital period regardless of how close or how far the object is from the central body.

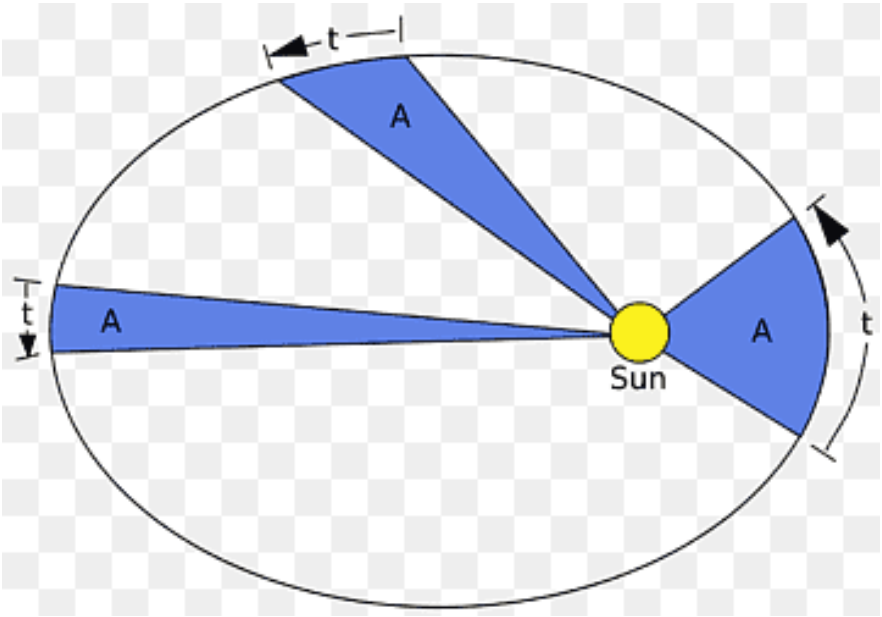


Figure 3: Visual of Kepler's second law showing equal areas swept out in equal periods of time regardless of proximity to the central body [51].

The proportional relationship of the third law can be derived from balancing the force of gravity with the centripetal force.

$$
\begin{aligned}
F_g &= F_c \\
\frac{GMm}{r^2} &= \frac{mv^2}{r} \\
\frac{GM}{r} &= v^2 \\
\frac{GM}{r} &= \left(\frac{2\pi r}{T}\right)^2 \\
T^2 &= \frac{4\pi^2 r^3}{GM} \\
T^2 &\propto r^3
\end{aligned}
\tag{5}
$$

This relationship, which is accurate for a perfectly circular orbit with an eccentricity of zero, extends to ellipses by having "r" be replaced with the semi-major axis "a". This gives us the proportional relationship given by Kepler's third law. While there are other factors to consider when discussing the motion of RSOs, such as atmospheric drag and solar radiation pressure, these fundamental laws generally describe the behavior of an RSO in orbit around Earth.

When discussing RSOs, there are several types of orbit to consider. The first of these is LEO. LEO objects orbit at altitudes between 200 km and 2000 km and their speeds are typically between 7 and 8 km/s. These orbits have low eccentricities, meaning they are almost circular in shape. These orbits are typically favored by EOS and communications satellites. In LEO, the effects of factors such as atmospheric drag and the oblateness of Earth on RSOs are more pronounced. These factors cause LEO orbits to decay faster. There are special types of LEO orbits, known as polar orbits and sun-synchronous orbits (SSO). Polar orbits, as the name suggests, involve satellites passing over the poles. In SSO orbits the orbital plane of the orbiting object maintains a fixed position relative to the Sun.

Medium Earth Orbit (MEO) extends from 2000 km to 35,786 km in altitude. MEO is favored by navigation satellites. In this type of orbit, the typical orbital speed is around 3 km/s and eccentricities can reach up to 0.7.

Beyond this, there are geostationary orbits and geosynchronous orbits (GSO). These orbits go beyond 35,786 km. Geosynchronous orbits vary in eccentricity, while geostationary orbits are

circular. These orbits are useful for communications and weather satellites.

There are also highly elliptical orbits (HEO) which have an eccentricity approaching 1. These orbits are useful for extended high-altitude observation missions and can be used as transfer orbits for interplanetary missions [52].

1.4 Orbital Elements

From Kepler's laws derive the Keplerian elements that characterize RSO orbits in detail and are essential for orbit estimation and RSO propagation. There are six of these essential elements in total. The first of these is the semi-major axis, as discussed in the previous section, which describes the size of the orbit and can be used to determine the orbital period via Kepler's third law. The duration of an RSO's orbit is important for determining potential windows of observation.

This is followed by the eccentricity, which describes the shape of the orbit. Higher eccentricities translate to more elliptical orbits. This pertains to Kepler's first law, which states that orbiting masses around a central body have elliptical orbits. The eccentricity of an object's orbit would affect motion and object brightness during observations.

Inclination describes the tilt of the orbit. More specifically, it describes the angle between the orbital plane of the orbiting body and the Earth's equatorial plane. A polar orbit would have an inclination of 90°, for example. The inclination determines where an object can be observed from a ground or space-based sensor. A space-based sensor would require an orbit that would give a line of sight to a target RSO.

The right ascension of the ascending node (RAAN) is the angle measured out from an imaginary axis that points in the direction of the Sun from the center of Earth at the time of the Vernal Equinox. This direction is also known as the First Point of Aries. RAAN can also be thought of as the rotation of the orbital plane around the polar axis of the Earth. After RAAN, there is the argument of perigee. This is the angle measured within the orbital plane from the ascending node to the perigee; the point in an orbiting object's orbit where it is nearest to the central body.

Finally, there is the true anomaly. This gives an orbiting body's current position in orbit.

Specifically, it is the angle measured out from the Argument of Perigee. This location changes with the motion of the orbiting body, and the rate of change is dynamic, based on Kepler's second law. The True Anomaly is extremely important for predicting object location and making RSO detections.

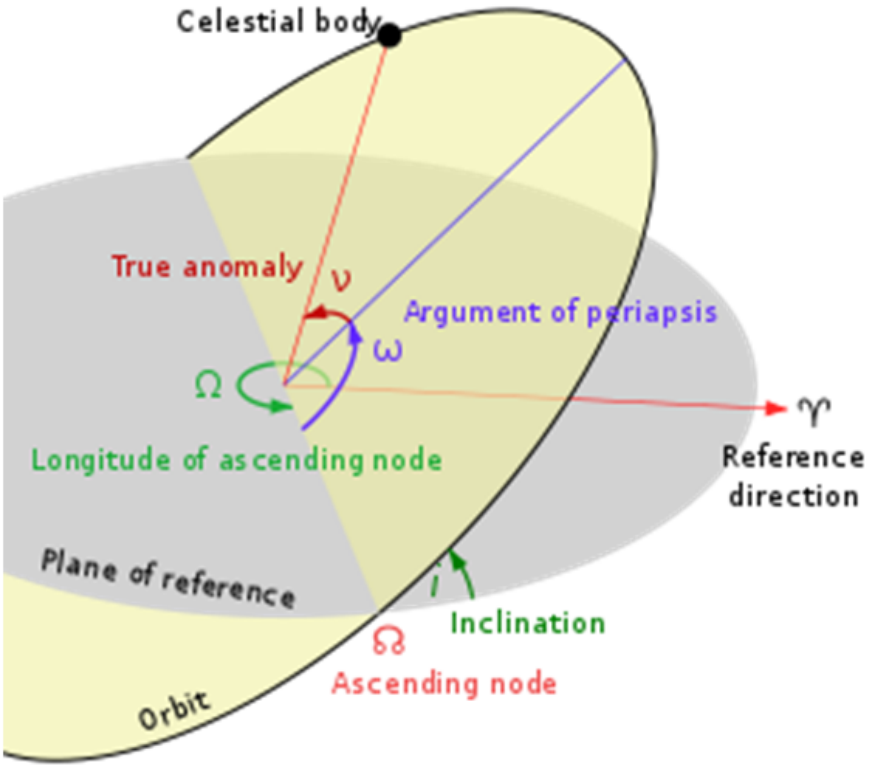


Figure 4: Diagram visualizing the Keplerian elements [56].

1.5 Detectability

Detectability of RSOs is closely related to the orbital elements introduced in the previous section. Aside from sensors having a physical line of sight to RSOs, detectability is also affected by illumination. The visual magnitude of an RSO depends on the solar phase angle, which is the angle between the RSO, the sensor, and the Sun. Determining the solar phase angle requires the position of the RSO, which is characterized by the six Keplerian elements. The true anomaly angle is particularly useful in this regard.

The solar phase angle determines how much of the RSO is illuminated and, by extension, the brightness of the object from the point of view of the sensor. With the solar phase angle, the visual

magnitude of the object can be calculated.

$$\Phi = \pi - \cos^{-1}\left[TargetPosition_{ECI} - HostPosition_{ECI}\right] \cdot Sun_{ECI} \tag{6}$$

This equation shows that the phase angle is based on the dot product of the distance of the target from the observer (in the Earth-centered inertial frame) with the position of the Sun.

$$F(\Phi) = \left(\frac{2}{3\pi^2}\right)\left[(\pi - \Phi)\cos\Phi + \sin\Phi\right] \tag{7}$$

$$m_{obj} = m_{sun} - 2.5log\left(\frac{\rho A_{target} F(\Phi)}{R^2}\right) \tag{8}$$

In the visual magnitude equation, ρ represents an RSO's reflectivity and A_{target} is the area of the RSO. This is particularly relevant when discussing the simulation of RSOs for detection in subsequent chapters of this thesis. A more detailed explanation of RSO visual magnitude can be found in [57] [44].

1.6 Outline

The content of this thesis is as follows: Part 1 explores the application of deep reinforcement learning for sensor tasking in SSA and encompasses chapters 2, 3, and 4. Chapter 2 provides background on the sensor tasking optimization problem in SSA, as well as an overview of optimization algorithms, followed by an overview of reinforcement learning (RL) and deep reinforcement learning (DRL) techniques. Chapter 3 covers the research into the application of deep reinforcement learning algorithms to simulated SSA scenarios on which this thesis is partly based. This topic is covered in Chapter 4, which includes a summary and proposed future work. Part 2 explores the application of Faster R-CNN to object detection for SSA and encompasses chapters 5, 6, and 7. Chapter 5 discusses background information on non-ML object detection algorithms, as well

as several other machine-learning algorithms outside of the R-CNN family before finally discussing R-CNNs. Chapter 6 then explores the research into the use of Faster R-CNN in SSA contexts on which this thesis is partly based. Chapter 7 gives a final summary and details future work plans for the topic. Chapter 8 ties together the two parts in a final examination of how the research discussed in this thesis contributes to the broader field of SSA.

Part 1: Sensor Tasking

This portion of the thesis is concerned with the use of DRL algorithms for sensor tasking optimization. Autonomous sensor tasking and the optimization of how we assign optical sensors for RSO catalog maintenance are of great interest as the number of RSOs continues to expand. With every new addition to the catalog, the enhanced risk of collisions in our overcrowded orbits necessitates the leveraging of more powerful sensor tasking optimizers. Given that DRL has proven to be effective in creating robust agents that can adapt behavior in dynamic environments. In this section the application of DRL is explored for autonomous sensor tasking in simulated environments involving a space-based optical sensor in order to demonstrate that novel DRL algorithms can be effectively applied to SSA and need to be studied further in the context of SSA and sensor tasking.

Chapter 2 provides a look at other optimization techniques that have contributed to sensor tasking, as well as an overview of how reinforcement learning works. It provides an explanation of the mathematical framework and how its combination with deep learning greatly expands the scope of what can be achieved with the approach.

Chapter 3 explores the applications of different DRL algorithms to two distinct SSA sensor tasking scenarios. It gives an explanation of algorithm performance metrics during training, the outcomes of training for each scenario, and an explanation of how these algorithms can be customized in their implementation and tailored to specific problems according to their complexity.

Chapter 4 concludes this section of the thesis by summarizing the DRL research conducted in Chapter 3. It also presents key findings and outlines potential avenues for future research and development. The results demonstrate the adaptive capability of DRL algorithms to explore environments and continue to improve and refine their approach to optimizing the use of sensors to detect and track RSOs.

2 Sensor Tasking Optimization Background

The SSA community has been researching the use of optimization algorithms as a means to create optimal sensor tasking schedules for space and ground-based observation instruments for decades. Given the overwhelming complexity of the sensor tasking problem, and the limited supply of sensors struggling to keep pace with an exponentially increasing surplus of RSOs, powerful computational optimization algorithms have become more necessary to produce effective scheduling assignments. Non-RL algorithms that have been used in sensor tasking applications generally fall into two categories: greedy algorithms, which are myopic and employ rudimentary techniques focused on exploiting the most immediately rewarding solutions, and evolutionary algorithms, which are more exploratory and exploit the principles of natural evolutionary processes in their approach. The latter algorithms are generally better at creating more effective long-term optimization solutions than greedy algorithms. However, this cannot always be guaranteed. It depends significantly on the nature of the optimization task. Some tasks lend themselves better to one type of approach over another.

While both of these approaches have their benefits and drawbacks, they both have shortcomings that a marriage between deep learning and RL, namely DRL, can overcome. DRL is a relatively new approach to computational optimization that has several distinct advantages. Deep reinforcement learning methods have shown great promise in overcoming the limitations of other optimization methods in the SSA domain for certain applications [14, 15, 16, 17]. In fact, deep learning methods in general have demonstrated evidence that they can overcome the "curse of dimensionality" in a variety of contexts. This is not to imply, however, that DRL can solve all forms of optimization challenges in SSA. It does have certain limitations and may not suit every problem, but it can potentially be used to find more optimal solutions to SSA problems that cannot be adequately optimized with more traditional methods.

While DRL is reasonably new in its present-day form, the mathematics behind DRL methods is not particularly novel. The mathematical basis of RL was established in the mid-20th century. It is only in recent decades, however, that these algorithms were merged with deep learning techniques to take advantage of neural network architecture. This combination provided the powerful function-

approximation framework required to approximate RL functions for very advanced optimization problems with large numbers of states and actions. The strength of this combination has only become more apparent in the last decade as more powerful hardware has become available to run large DRL models.

2.1 Greedy Algorithms

Various greedy algorithms have been used in SSA contexts for some time. These types of optimization algorithms are designed to maximize an immediate reward. This means that they will attempt to find a globally optimal solution by always choosing a locally optimal solution at every step. This may be useful in some limited SSA contexts, but ultimately is not a considerably reliable approach to SSA scheduling optimization.

Hill Climbing is one such example of a myopic algorithm. As the name implies, the algorithm will follow the gradient of the function representing the quality of all possible solutions which, in the sensor tasking context, would be schedules of sensor assignments. The function that scores the schedules is the fitness function. The algorithm will follow the gradient until it reaches an optimum. The algorithm will generally begin its search from some random initialization point, although these can be intelligently chosen as well. If there is only one local optimum (i.e. the problem is convex) then this type of algorithm can be expected to perform well and find the global maximum or minimum. However, if there is more than one maximum or minimum point, the algorithm will likely get stuck around a local optimum [18].

Consequently, greedy algorithms are generally insufficient for handling problems that are not convex and can generally offer only suboptimal solutions to SSA problems. There are of course variations on greedy algorithms that can make them more effective at handling non-convex problems. In the case of Hill Climbing, a variation known as Simulated Annealing adds noise to the algorithm in order to encourage exploration beyond local optima but this would still have considerable limitations in more complex problems. That is not to suggest that these algorithms are useless in SSA contexts. Variants such as Tabu search have seen success in satellite imaging scheduling, for instance, due to a few enhancements on the basic Hill Climbing algorithm [19]. These enhancements include a store of recently visited solutions in order to prevent the repetition of inadequate decisions. The ability

to store a memory of past experiences to influence future decisions is a valuable addition, and this is a concept explored by DRL algorithms as well to much greater effect.

To appropriately highlight the limitations of greedy schedulers in SSA contexts, it is worthwhile to discuss the use of these algorithms in the SSN. The SSN uses greedy tasker programs for sensor assignments in the real world. A main tasker takes all current cataloged RSOs as input and, based on the orbit and characteristics of the RSOs, as well as the specific parameters of the sensors in the SSN, calculates the probability of detection for each RSO. This allows the tasker to assign sensors to particular RSOs. RSOs are assigned to various sensors based on their detectability probability and a secondary greedy scheduler at the site of each sensor determines when to track detectable RSOs based on opportunity and priority.

There are several limitations with this process. Since observation scheduling is handled independently for each sensor, schedules cannot be globally optimized since the scheduling assignments of other sensors are not shared. It is also impossible to create automated responses to tasking requests as they come in. In other words, dynamic scheduling is impossible. Scalability is also a considerable challenge. The addition of new sensors, including sensors that can track smaller and smaller objects, the number of sensors to task and the number of objects to track will continue to grow [53].

There are many factors to consider when attempting to schedule RSO observations. A lack of dynamic scheduling ability means that greedy algorithms cannot account for sudden changes in tracking priorities. They also cannot handle situations where sensors become unavailable due to technical issues.

2.2 Evolutionary Algorithms

Evolutionary algorithms, such as genetic algorithms, are based on mechanisms of biological evolution and other types of biological mechanisms, like swarm intelligence. In the case of genetic algorithms, for instance, the algorithm is presented with an initial random population of solutions that are then ranked according to an objective or fitness function. This function produces fitness values for each solution. Based on these fitness values, the lowest-ranking solutions are discarded,

and the surviving solutions form the next generation of the population – the parent population – which will then be used to produce additional offspring. The offspring solutions will then be used to create more solutions, often with some mutation added. The process will repeat until an optimal solution is discovered and the fitness value is maximized. Evidently, this process mimics natural selection.

There are similar algorithms that exploit other evolutionary principles, such as Ant Colony Optimization and Particle Swarm Optimization. These exploit the principles of swarm and collective intelligence, with a large number of agents exploring a search space of candidate solutions to find an ideal candidate.

Evolutionary strategies are best suited to solving global optimization problems involving large search spaces and large combinatorial optimization problems like the Traveling Salesman Problem [20]. They are more complex and computationally expensive than greedy algorithms but can generally be expected to produce better results because they are more capable of generating reliable long-term strategies.

2.3 Scheduling Optimization in Space Situational Awareness

In reality, sensor tasking optimization is often performed manually with the assistance of scheduling algorithms. The authors of [21] list several manual scheduling programs that are used, such as the Systems Tool Kit (STK) Scheduler. The STK Scheduler makes use of different types of optimization algorithms for scheduling, such as those of the greedy variety, but allows users to manipulate generated schedules.

Many optimization algorithms have seen applications in sensor tasking. Before DRL methods became widely accessible, various researchers attempted to explore the potential benefits of both greedy optimization and more sophisticated evolutionary methods. For example, the authors of [22] explore a genetic algorithm called the Non-dominated Sorting Genetic Algorithm II (NSGA-2) to optimize the scheduling of observations taken by Earth-observing satellites. This is a type of multi-objective optimization algorithm, based on metaheuristics, that follows a modified genetic algorithm approach to optimization. The authors of [23] compare greedy algorithms with a genetic algorithm

as solutions for scheduling Earth-observing satellite observations. In their particular simulations, they discovered that Hill Climbing and Simulated Annealing surprisingly outperformed their genetic algorithm. From this they concluded that the Earth-observing satellite (EOS) problem seems to favor an optimization approach based more on exploitation than exploration, demonstrating that the effectiveness of various types of optimization strategies can be significantly problem-dependent. In [24], the authors compare the greedy algorithms Priority Dispatch and Look Ahead with a genetic algorithm in another EOS problem, discovering that the genetic algorithm was able to perform 3% better on average than the Look Ahead algorithm, which performed better than Priority Dispatch.

The sensor tasking problem in SSA continues to challenge researchers and as such will continue to require increasingly powerful optimization techniques and increased automation. Ideally, sensor tasking optimization algorithms should be forward-looking so that they develop long-term scheduling solutions, be convenient to implement, and be self-correcting. While evolutionary algorithm approaches are better at generating long-term solutions and are self-correcting to an extent, they are not always convenient to implement. Determining a fitness function by which to evaluate candidate solutions can be a significant challenge depending on the complexity of the problem. Designing a good fitness function often requires considerable knowledge of the problem domain. This is an area where RL has a distinct advantage.

2.4 Reinforcement Learning

RL is one of the three machine learning paradigms and is host to a range of optimization algorithms that follow the Markov Decision Process (MDP) formulation. In the MDP formulation, a state is presented to an agent, the agent chooses an action to take, an outcome is achieved in the form of some reward, and the state changes. This process happens recursively throughout the runtime of an RL environment.

The mathematical function that guides the agent's decision-making process is called a policy. How the policy is developed and improved upon will depend on the type of algorithm used. The main goal of reinforcement learning, however, is to find an optimal policy that maximizes the expected cumulative reward of an agent through dynamic interaction with the environment. A policy function, often denoted by π, is a probability distribution of actions that an agent can take

given that it is presented with some state. There are deterministic policies, where a state is directly mapped to an action. There are also stochastic policies where, for every state, there may be multiple actions to take with a certain probability for each.

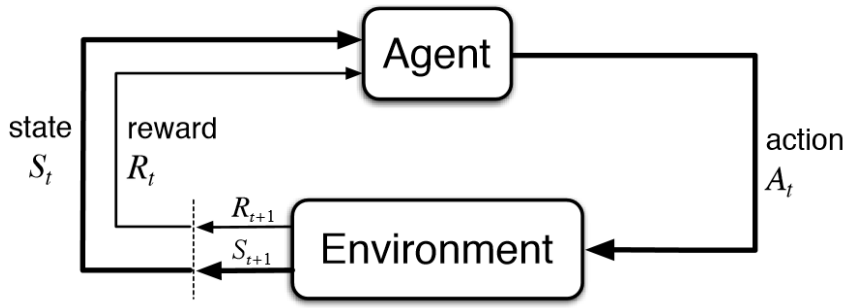


Figure 5: Diagram of Markov Decision Process [25].

2.5 Value-based Policy Optimization

There are different approaches to policy optimization in RL. The first of these is the value-based approach, as seen in the Q-learning or SARSA methods of RL. These algorithms fall under the umbrella of temporal difference (TD) learning. In the value-based approach, an agent will develop its behavior policy based on either estimating the value of taking a specific action while being in a specific state or estimating the value of being in a specific state, depending on the type of algorithm. Some algorithms use state-value functions, defined by Equation 9. These measure how valuable it is for an agent to be in a particular state at each time step and then follow the current policy from there.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\Sigma_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s] \tag{9}$$

Others use action-value functions that give Q-values, or "quality values", defined by Equation 10. Q-values measure the value of being in a particular state and choosing a particular action, then proceeding to act according to the current policy. The moniker of "quality value" comes from the notion that Q-value functions assess the quality of an action taken when presented with some state.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\Sigma_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{10}$$

In either case, the policy guides the agent towards either states or state-action pairs that produce the highest values or Q-values at given times. This implies that the initial policy is often a greedy policy or a variant of a greedy policy. Value functions give the expected discounted future return that an agent will receive when starting in some state and then acting according to some policy from that point on. The expected return from taking specific actions is the sum of the discounted rewards generated from each of those actions. The discounted future return is defined by Equation 11.

$$G_t = \Sigma_{k=0}^{\infty}\gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{11}$$

These rewards are collected for each time step beyond the initial state and summed using an applied discount factor γ which will be some value between 0 and 1. This discount factor decides the importance of future events by placing more mathematical significance on more immediate rewards. An agent will seek to maximize the value as it navigates an environment by choosing certain actions. Because of the mathematical nature of value functions and the Bellman Equation [26] on which they are based, they have a dependence on the value function of the next state or state-action pair. This process of improving value function estimates by looking ahead to the next time step is called bootstrapping and is the core mechanism behind TD learning. So, as the agent navigates the environment and encounters states, the value functions of each state are updated iteratively using increasingly accurate estimates of the value of each succeeding state with each iteration. This in turn implicitly updates the policy of the agent because the agent will continue to choose actions that lead it to states and state-action pairs with the highest values. Therefore, as the values are refined and improved, the policy will change accordingly to accommodate these updated state values. Additionally, the policy will generally be updated after every time step for fast convergence.

2.6 Policy-based Policy Optimization

There are also policy-based methods in which an agent learns a policy directly, such as in policy-gradient approaches based on the REINFORCE algorithm [25]. The agent will take actions under a policy when presented with some state and these actions will earn the agent a numerical reward. At the end of an episode of exploration, the agent will take the rewards and compute the discounted future return at each time step. This future return is then used to update the agent's policy and increase the probability of taking "good" actions while decreasing the probability of taking "bad" actions. We also have the combination of policy-based and value-based methods, known as the Actor-Critic approach.

Policy-based methods have some advantages over value-based methods, in the sense that they can handle agents making either discrete or continuous actions. Additionally, they can learn stochastic policies as well as deterministic policies. Value-based methods are relegated to being capable of learning only deterministic policies. Another shortcoming of value-based methods is that sometimes the value function may be difficult to learn in certain environments, making it challenging to implicitly update the policy. In some instances, it may be easier to simply learn the policy directly.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\quad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Figure 6: REINFORCE algorithm [25].

2.7 On-Policy and Off-Policy Optimization

As previously mentioned, the policy determines what action an agent will take when it finds itself in a particular state. It is a probability distribution indicating the likelihood of an agent taking a certain action. Depending on the algorithm, the agent may take either an on-policy or an off-policy approach to learning. With an on-policy algorithm, the policy that is being evaluated and updated

is the same as the policy that governs the agent's decision-making, known as the target policy. In off-policy models the behavior and target policies can differ, leading to greater stochasticity and therefore greater exploration of an environment. Q-learning makes use of off-policy learning through its ϵ-greedy approach to policy. Most of the time the agent will follow the target policy. This is the policy that is being evaluated and updated iteratively. Sometimes, however, the agent may act according to a different policy and explore actions outside of the target policy. Algorithms of this type can store past experiences in an experience replay buffer, which the agent will sample from time to time to aid in learning. Because of this, off-policy value-based algorithms are generally more sample-efficient than other reinforcement learning approaches.

2.8 Deep Reinforcement Learning

In DRL, deep neural networks are used to solve the functions of the RL model through gradient calculations and backpropagation. There are model-based and model-free DRL methods but this thesis is only concerned with model-free methods in which an agent explores an environment with no prior knowledge of the space. These are the most studied DRL algorithms.

DRL algorithms have proven to be powerful tools of optimization in various real-world applications because not only are they dynamic and adaptive, but they can also take in high-dimensional data such as images due to their use of neural networks. Other, more traditional optimization algorithms cannot do this. In recent years, DRL agents have been trained to beat human players and achieve extremely high scores in classic video games such as Pong and Space Invaders by analyzing frames from these games [27]. They have even been able to learn optimal tactics in more modern video games. For example, in 2019, several DRL agents managed to beat a world-champion human team in a live match of Dota 2. These agents were trained for approximately 10 months [28]. DRL has also been used in control tasks like autonomous driving and as a means of teaching robots how to navigate environments in search of a specific goal.

Artificial neural networks process information using layers of nodes with assigned weights that amplify or suppress features of the input data. In general, neural networks seek to minimize a loss function, which is the error in the network's best estimate of how to correctly interpret data compared to the real truth data. In order to minimize this error, the network will "backpropagate",

where it will adjust the weights of each node in a direction that makes future estimates more correctly fit the reality of the data input. In the context of DRL, there is no data input in the same way as there would be in supervised or unsupervised learning models. Instead, there is an environment that gives feedback in the form of a reward. In DRL, backpropagation adjusts the weights of the network nodes to alter an agent's behavior in service of reaching a higher reward and models the value or policy functions of the DRL agent. This combination offers a promising approach for automating and optimizing many different kinds of sequential decision-making and control tasks.

As is the case with regular RL, model-free DRL algorithms can generally be divided into value-based methods and policy-based methods, or a combination of the two. Just as in regular RL, the value-based methods use a type of value function that estimates the expected future reward an agent can achieve in a specific state or state-action pair in order to find an optimal policy that maximizes a total reward. In DRL, the value function is approximated by a neural network. This is because neural networks are universal function approximators. Policy-based methods will directly determine an optimal policy that maximizes a total reward through parameterization. In such algorithms, the policy itself will be approximated by a neural network. There are also algorithms that combine the value and policy-based approaches called Actor-Critic algorithms that use a "critic" value function to evaluate a policy function known as the "actor," both of which will be approximated by neural networks. Algorithms from these categories can also be either on-policy or off-policy models.

2.9 Policy-Gradient Methods

Policy-gradient algorithms are a subset of policy-based optimization. As stated previously, the policy function is a probability distribution that maps states to actions. In DRL, the policy is approximated and parameterized by a deep neural network. Actions are then selected according to a softmax activation function defined by Equation 12.

$$\pi(a|s,\theta) = \frac{\exp h(s,a,\theta)}{\Sigma_b \exp(h(s,b,\theta))} \tag{12}$$

With each training episode rollout, the algorithm will keep track of states, rewards, and actions. This information will be used to calculate the discounted future return at each time step. Using these returns for weights and the actions taken as labels, the neural network will perform backpropagation to update the policy. This process is repeated until the agent learns an optimal policy. The parameters of the neural network are updated according to Equation 13.

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{13}$$

In this equation, $J(\theta)$ is the performance measure function. The value of $\nabla J(\theta_t)$ is given by the proportionality relation of Equation 14.

$$\nabla J(\theta_t) \propto \Sigma_s \mu(s) \Sigma_a q_\pi(s, a) \nabla \pi(a|s, \theta) \tag{14}$$

The μ function is the visitation rate, which is the fraction of time a state is visited under a policy. The q term is a value for state-action pairs. This is of course followed by the gradient of the policy itself. This can, however, be expressed as an expectation value in the form of Equation 15.

$$\nabla J(\theta_t) = \mathbb{E}_\pi \left[G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \tag{15}$$

The G_t term represents the discounted future return. At the same time, the gradient of the policy over the policy is a vector in policy space that represents a direction that maximizes the chance for action A_t to be repeated. When multiplied by the scalar expected future return, this becomes a vector that increases the probability of choosing actions with high expected future returns. This is how the parameters, and therefore the weights of the neural network approximating the policy, are updated in a way that improves the agent's policy.

There are a couple of key shortcomings to this method. Unlike off-policy value-based methods, the agent does not store past experiences in memory. At the beginning of each episode rollout of an

environment, the agent's memory is effectively reset. Additionally, there is still a large probability of selecting any different action in a given state. This means that there will be significant variation between episodes.

Fortunately, these shortcomings can be addressed with rather simple solutions. In order to address sample inefficiency, the agent can be allowed to run through a batch of episodes before updating its policy so that it has the opportunity to visit states more than once. This batch size can be customized as well. To tackle the problem of high variance between episodes, the rewards can be scaled to some appropriate baseline. The most appropriate baseline to use is the average reward from each episode. With these additions, policy gradient methods can handle extremely complex problems that other DRL methods cannot.

2.10 Examples of Deep Reinforcement Learning Algorithms

A deep Q network (DQN) [29] is a value-based, off-policy DRL model. This is one of the DRL algorithms that was used for the research discussed in Chapter 3. In Q-learning models like DQN, there is a state-action value function, represented by the recursive Q(s, a) function (where s is the state input and a is the action input). The output of the Q function is known as the Q-value which is a measure of how good it is to be in a given state "s" and take an action "a", then continue under the policy. In Q-learning the agent will seek to learn a particular set of Q values that will lead to a maximum reward, and therefore an optimal outcome. An agent will take an action based on a behavior policy which is often ϵ-greedy. For context, an ϵ-greedy policy is one in which the agent follows the current target policy with a probability of 1-ϵ and sometimes takes a random action with probability ϵ. This in turn forces the environment into a new state and the agent is given a reward for its action. At this point, the Q value is calculated using the Bellman Equation (Equation 16) based on the sum of the reward for that state-action pair.

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a) \tag{16}$$

The Q value is given by the present reward combined with the discounted max Q-value of the next

state. The difference between this Q-value and the expected Q-value is known as the TD error. The expected Q value is then updated by adding on the TD error multiplied by the learning rate. The learning rate is the factor that dictates how much the Q-values will change per update, which will change how quickly the model converges to a solution. The learning rate is an adjustable parameter in all DRL algorithms. Choosing an appropriate learning rate is a careful balance, however, and is largely dependent on the environment and the algorithm. Some algorithms are more robust, but generally, the performance of a DRL algorithm can be greatly affected by hyperparameter tuning.

DRL methods generally require a tremendous sample of instances of a task to be able to learn to perform a task well, even for a relatively simple task. This only increases with the complexity of the task. This is a particular problem for on-policy algorithms that, unlike off-policy algorithms, cannot reuse past experiences to learn a task. Off-policy DRL algorithms can store past experiences in replay buffers. However, some on-policy models support parallel training of multiple agents, or "workers," on one environment.

An example of a popular on-policy DRL algorithm is Advantage Actor-Critic (A2C). This is one of the algorithms that was used for the research discussed in Chapter 3. As the name suggests, it is an on-policy policy-based approach that uses the actor-critic framework. This means that it is a hybrid of the value and policy-based approaches. It is still a policy gradient model, but it uses a secondary "critic" network that approximates the value function for each time step to evaluate the "actor" network's policy estimation. In a way, the actor-critic framework bears some similarity to Generative Adversarial Networks (GANs), in the sense that there are two networks working together to solve a problem. "Advantage" refers to the "advantage" function which is used in A2C to stabilize learning. The Advantage function can generally be defined as shown in Equation 17.

$$A(s, a) = Q(s, a) - V(s) \tag{17}$$

This quantity is the difference between the action-value and state-value functions. This tells the agent how much better or worse it is to be in a state and take a certain action rather than being in said state and taking the average action. In A2C however, there is no calculation of Q(s,

a) performed. This means that it must be approximated using another quantity. Fortunately, the Advantage function can be approximated by the TD error. In this case, the Q(s, a) term is replaced with $r_t + \gamma V(s_{t+1})$.

2.11 Advantages of Deep Reinforcement Learning

DRL methods are very powerful for optimization, including scheduling optimization. However, they are not advantageous in every type of optimization problem. They work best in sequential decision-making and control tasks, where the temporal nature of these tasks lends itself well to the MDP formulation. This is useful in scheduling optimization because scheduling involves deciding on a sequence of actions over time.

Through the use of deep neural networks, DRL algorithms can handle high-dimensional data and complex environments with large state and action spaces, particularly those involving sequential decision-making tasks that can be formulated as an MDP. This puts evolutionary strategies at a distinct disadvantage. The ability to handle high-dimensional data means that agents can learn from sensory inputs like images. In optical sensor tasking contexts this has the potential to be extremely useful.

One of the most advantageous aspects of DRL is the fact that agents learn dynamically from interaction with the environment, whereas other optimization techniques attempt to generate or find an ideal candidate solution from a starting pool of solutions. This permits exploration through the use of continuous actions, rather than only discrete. This is especially useful when attempting to optimize control tasks. Certain DRL algorithms, like A2C, can handle continuous action spaces, meaning that the actions an agent can take exist within a continuous range. If a task requires finer movement and control, this is an extremely useful characteristic.

DRL agents are also more generalizable in theory, with agents sometimes able to apply learned policies to similar environments with little to no additional training. This is something that evolutionary algorithms cannot do. However, this is still the subject of research and not something that can be effectively relied upon. However, one method of enhancing generalizability in DRL is training agents in a variety of similar environments. To draw a comparison with other methods of

deep learning, such as supervised learning, then a single instance of an environment is similar in a sense to a single data point. A supervised learning model would fail to generalize if trained on a single data point, but with an entire dataset and trained for many epochs it could generalize very well.

Another significant advantage of DRL is the ability to store memories of actions and policies that led to negative rewards. Genetic algorithms, by comparison, discard poor candidate solutions. DRL methods are also adaptable, with agents able to react to changes in the environment as they navigate said environment. Overall, DRL methods are extremely convenient and intuitive methods of optimization and are also less difficult to implement effectively than many other optimization approaches. However, even with all these advantages, convergence to the global optimum cannot always be guaranteed. There are many tunable hyperparameters available for each algorithm that can drastically alter a DRL agent's performance. Network depth and size can also greatly impact the agent's ability to learn an environment well. Sometimes, the random initialization values of the network parameters can have a profound impact on training. It is for this reason that DRL researchers often stick with particular seed initialization values in order to replicate results.

In terms of hardware implementation, DRL algorithms can be run on Nvidia GPUs and take advantage of GPU parallel processing. This can significantly improve the speed with which agents are trained and is a feature generally not shared by either evolutionary algorithms or greedy algorithms. These are generally relegated to running on CPU hardware only.

2.12 Previous work on DRL on SSA

Some efforts have already been made to incorporate DRL into SSA contexts for the purpose of sensor tasking optimization. The authors of [14] used a DRL agent based on the Proximal Policy Optimization (PPO) algorithm approach to minimize the positional uncertainty, quantified by their mean trace covariance, of simulated Geostationary Equatorial Orbit (GEO) RSOs that were propagated using Simplified General Perturbations 4 (SGP4). A custom DRL environment was created in Python using the OpenAI Gym API that would propagate a number of RSOs, as well as a space-based sensor with a small 4°x4° field of view (FOV) to track RSOs. The sensor's field of regard (FOR) consisted of 360° in azimuth and 104° degrees in elevation. This FOR was discretized

into 2340 FOV-sized windows corresponding to 2340 sensor-pointing directions in order to limit the size of the action space. Two scenarios with different numbers of simulated RSOs are tested and shown to have better results in minimizing positional uncertainty than myopic greedy algorithms. Authors of [15] use the same scenario and algorithm but use a ground-based observation platform rather than a space-based one. Similarly, the authors of [17] used a Double DQN to minimize the positional uncertainty of LEO RSOs from a ground-based sensor. The authors of [16] attempt to solve a different problem. They attempt to use RL for catalog maintenance, maximizing the length of time that each simulated RSO is viewed and attempting to maximize the number of unique RSOs detected. They compare a greedy algorithm and Ant Colony System (ACS) algorithm against a Distributed Q-Learning (DQL) approach with multiple agents. Given the nature of the problem, however, the ACS algorithm performed the best. This is because evolutionary algorithms are better suited to large-scale combinatorial problems where a large area must be explored in a non-sequential way.

3 Scheduling Optimization

3.1 Methodology

The research goal within this portion of this thesis was to explore DRL in optical sensor tasking applications. This required a simulator to propagate RSOs that would be detected or tracked, as well as a satellite with an onboard optical sensor that could be tasked to make observations within the RSO search space. The simulator used for this research is known as the Space Based Optical Image Simulator (SBOIS) and was developed by the York University Nanosatellite Laboratory in MATLAB [30]. The name is somewhat of a misnomer, however, as the simulator can also perform ground-based observations. The simulator uses SGP4 propagation, which is the specific mathematical model that calculates and updates the orbital state vectors of RSOs. It incorporates the effects of Earth oblateness with J2 perturbations, as well as atmospheric drag effects, perturbation effects from nearby bodies, and other perturbation effects. Cataloged RSOs can be propagated from two-line element (TLE) sets and sensor parameters of the observer satellite or ground observatory can be adjusted as desired. Further information regarding how the simulator calculates flux from various sources, RSO brightness, and how orbit propagation is handled can be found in [30].

While the simulator is built in MATLAB, the DRL training was facilitated in a Python environment using the MATLAB Engine API to run the simulator and the DRL libraries OpenAI Gym and Stable-Baselines3 (SB3) for agent training. MATLAB Engine API allows users to run MATLAB scripts in Python, while OpenAI Gym provides a framework for developing DRL environments. Using Python class structures, users can define the parameters of the space that the DRL agent explores, such as the actions it can take and the reward it will receive. As each environment built in the Gym framework is coded as a Python class, each instance of the environment is therefore a Python object. Within the framework, a user must define an initialization function within the environment class, as well as a step function that encodes agent behavior at each time step, followed by a reset function that returns the environment to some initial state. SB3 is a library that provides a number of DRL algorithms that can be used with the Gym framework and have many different configurable hyperparameters that affect training stability.

There were two scheduling optimization scenarios studied in this thesis. For both of these scenarios, the observer satellite chosen was the Canadian Space Agency's Cascade SmallSat and Ionospheric Polar Explorer (CASSIOPE) satellite. Specifically, the sensor used is CASSIOPE's Fast Auroral Imager (FAI) payload. This optical payload was originally designed to observe auroral emissions in the 630-1100 nanometer wavelength range and sports a very wide FOV. However, for this research, CASSIOPE's FAI payload parameters were discarded in favor of a modified version of Near Earth Object Surveillance Satellite (NEOSSat)'s sensor parameters in an effort to create a custom satellite configuration. The normal NEOSSat FOV of 0.85x0.85 degrees was changed to 10x10 degrees, with the per-pixel FOV also adjusted to accommodate this modification. This is because the default FOV was too small, and FOVs smaller than 10x10 caused technical problems with SBOIS. Thus, 10x10 degrees was adopted as the smallest FOV that was compatible with SBOIS. The reason CASSIOPE was chosen to provide the physical satellite was because there was already publicly available ephemeris data for it, which was very convenient. The physical movement restrictions of neither the NEOSSat nor the CASSIOPE sensors were considered in this research, as the scope of this research was more focused on the exploration of the concept of using DRL in SSA sensor tasking, rather than creating extremely realistic scenarios.

3.2 Sensor Slew Path Optimization

The first scheduling optimization scenario considered in this work focused on finding an optimal slew path for the satellite's sensor that would maximize RSO detections. The setup was such that the sensor would slew in the azimuthal direction from 0° to 300° with respect to the body of the satellite. The ability of the sensor to be able to realistically make such movements was not considered for this particular toy problem scenario. Therefore, there are no limitations on the degrees of freedom of the sensor, no power costs are considered, and no spacecraft attitude maneuvers occur. The sensor effectively moves as if it is not bound to a rigid frame.

The sensor's elevation was the aspect of motion controlled by a DRL agent. The implementation of this optimization approach was performed using the SB3 A2C algorithm due to the algorithm's simplicity, and the effectiveness of the hybrid actor-critic framework and policy gradient methods overall. In this scenario, the region of the sky in which the sensor slews is conceptualized as a grid

of FOV-sized windows. The agent must choose from one of three discrete actions; these being to increase elevation by 10°, thereby centering the sensor's FOV on a new grid window at a higher elevation, or to do nothing and maintain its elevation, or to decrease elevation by 10°, thereby centering the sensor's FOV on a new grid window at a lower elevation. It would have to make these movements while continually slewing in the azimuthal plane in 10° increments from 0° to 300° for a total period of observation of one minute. This one-minute-long scenario would be split into 30 time steps with 2-second step sizes. The length of the scenario is short but was chosen because, at this stage in the research, the simulator was only capable of handling smaller time increments without error. This was later rectified for the implementation of the second toy problem scenario.

As stated previously, the observation region is conceptualized as a 4 π-sr sphere of 10° x 10° windows. Azimuthal angles beyond 300° are excluded, as the sensor will not slew past this point in the time frame allotted. Elevation angles below 0° were excluded in order to keep the FOR from being far too large for a preliminary exploration of DRL in this type of scenario. A larger observation space would make the training duration and environment runtime much longer, taking up time that would be necessary to ensure the environment class was encoded properly in the first place, or that the scenario had been properly set up in the MATLAB script. The FOR is still quite large, with a 300° x 90° region of space for the sensor to traverse.

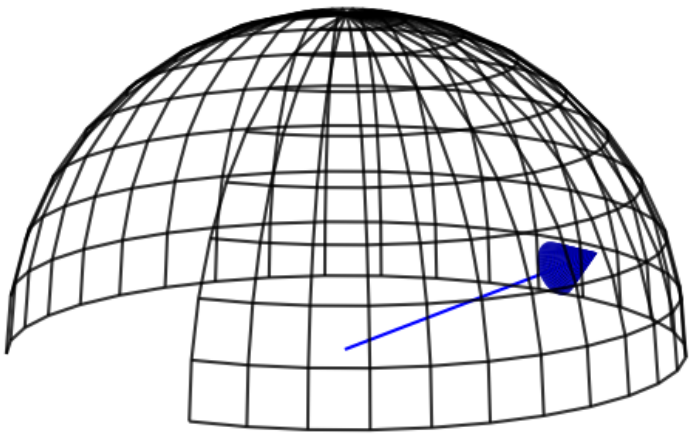


Figure 7: Diagram showing how the field of regard is conceptualized, with an arrow to illustrate the sensor pointing direction. Each square in the grid represents a unique discrete pointing direction equal to the size of the sensor's FOV.

In terms of how the reward system works, the agent controlling the sensor receives a reward proportional to the unique RSO count within the FOV at each time step. 10,000 RSOs of varying orbital radii were randomly selected from the SSN catalog and propagated in SBOIS for this scenario. RSOs in this scenario were all simulated as spherical objects for the sake of simplicity, all with diameters of 10 meters. Two reward schemes were tested using the same scenario. For the first reward type, the reward the agent received at each time step was a normalized RSO count. It is generally best to scale rewards in DRL problems on a scale of -1 to 1, otherwise it can have an impact on the training stability of algorithms. Beyond this, the agent would receive punishment for choosing any action that led to exploration outside of the prescribed FOR. Any move that placed the agent outside of the FOR would immediately terminate the episode and the agent would receive a reward of 0 for that action. The incentive for the agent to stay within the specified boundaries during training is shown in Equation 19.

$$r(a) = \begin{cases} \frac{\text{RSOs Observed}}{\text{RSOs Simulated}} & \text{if } a \in \text{FOR} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

For the second reward type, the agent would receive the same normalized RSO count. However, it would only receive this reward if, at each time step, it received a larger RSO count than the previous count. Otherwise, it would receive a reward of 0. The agent would also receive 0 for going out of bounds, just as before.

3.3 Advantage Actor-Critic

For one-dimensional data, the A2C algorithm from SB3 uses a fully connected network of two layers by default, with 64 units per layer. The A2C algorithm comes with a large variety of tunable hyperparameters and settings that can significantly impact the agent's training stability and performance. Users can choose between a multilayer perceptron policy model type or a CNN policy model type depending on whether they are working with image data or not. The learning rate - the factor that scales adjustments to network weights during backpropagation - can be chosen by the user. The default for A2C is 0.0007 and this proved to work well for the first SSA toy problem discussed in this chapter.

Users can also choose how many steps an agent takes in the environment before updating the policy. This is where users can essentially define the batch size for training. This value was set to be equal to the total number of steps within the first scenario environment, or equivalent to one whole episode. One advantage of SB3 is that it allows users to train agents in parallel on multiple environments simultaneously which can speed up and improve training results, provided the user has capable hardware. In this case, the batch size would be the product of the number of steps ("n_steps") with the number of environment instances running in parallel ("n_env"). Since episodes for the slew path scenario are one minute in length with two-second time steps, there are 30 steps per episode.

Users can also choose the discount factor to apply to future rewards when the algorithm calculates the discounted future returns of states. By default, this value is 0.99 and, as this is a standard value, there was no reason to change it for the purposes of this research. Beyond this, there are a plethora of other tweaks that users can make to the algorithm that affect training performance, as well as what seed initialization values to choose when initializing the algorithm, and what type of device (GPU or CPU) users wish to run it on. Table 1 shows the hyperparameter values chosen in this training scenario

Hyperparameter	Setting
policy	“MlpPolicy”
env	env
learning_rate	0.0007
n_steps	30
gamma	0.99
gae_lambda	1.0
ent_coef	0.0
vf_coef	0.5
max_grad_norm	0.5
rms_prop_eps	1e-05
use_rms_prop	True
use_sde	False
sde_sample_freq	-1
rollout_buffer_class	None
rollout_buffer_kwargs	None
normalize_advantage	False
stats_window_size	100
tensorboard_log	“file\path”
policy_kwargs	None
verbose	1
seed	None
device	“auto”
_init_setup_model	True

Table 1: A2C hyperparameter configuration for slew path optimization scenario.

3.4 Slew Path Optimization Results - Reward Type 1

For the SSA scenario, the agent was trained for 300,000 time steps, or 10,000 episodes, with a simple reward scheme that rewarded the agent with the scaled observed RSO count for each FOV window as it moved across the sky. The RSO count was scaled out of 10,000, as this was the number of unique RSOs simulated at the time. This was done because, as stated earlier, scaling rewards between -1 and 1 is generally conducive to greater stability when training a DRL agent. The average reward of the agent over time is shown in Figure 8. The average reward per episode is a rolling average.

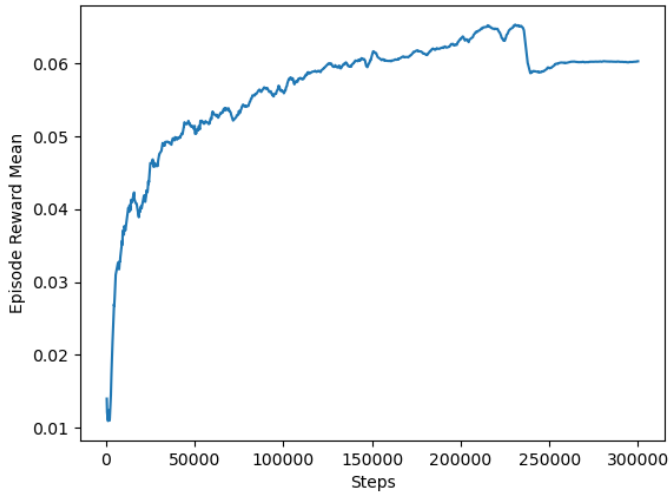


Figure 8: Agent average reward plotted over the course of 300,000 time steps. This is the average reward per episode, which steadily increases as the agent exploits better actions and earns higher and higher rewards.

The initial average reward was very low while the agent was still learning to stay within the boundaries of the FOR. There are many fluctuations, indicating that the agent continued to attempt new strategies throughout the training process. The length of each episode can be seen increasing over time in Figure 9.

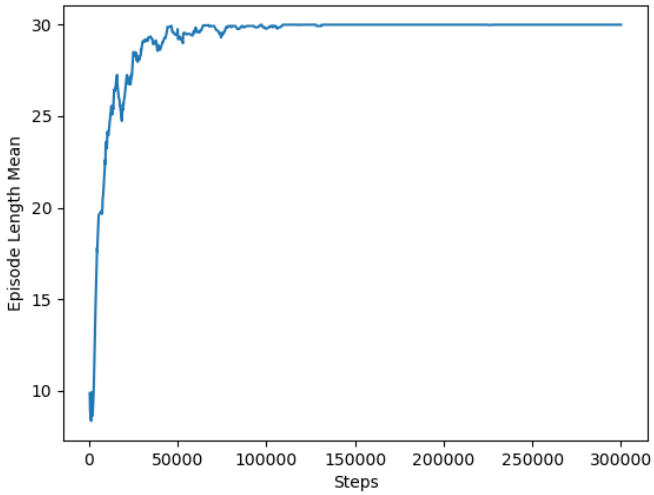


Figure 9: Average episode length plotted over the course of 300,000 time steps. This is the average length of an episode over time. The average length of an episode increases as the agent learns to stay within the boundaries of the FOR, thus leading to longer episodes and eventually full length episodes.

These metrics were the primary indicators that the agent was learning how to navigate the environment in better ways. Beyond that, however, there were other useful (although slightly less reliable) indicators for assessing the performance of the agent. The first of these are the actor and critic loss functions. These are depicted in 10 and 11, respectively.

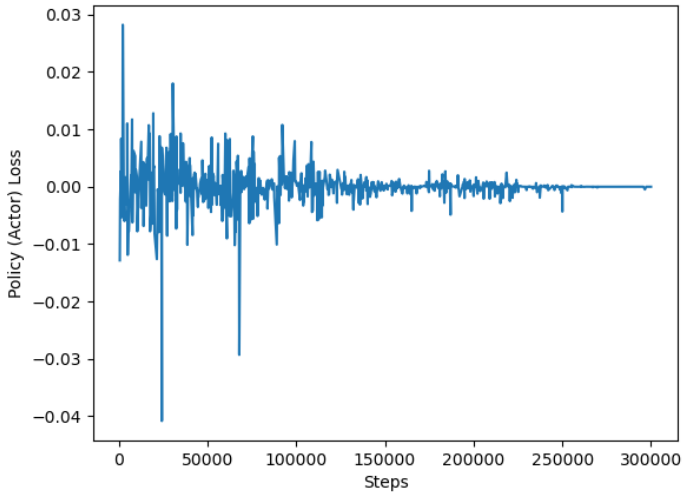


Figure 10: Actor/policy loss plotted over time. This indicates how much the policy is improving over time based on the Advantage function. In conjunction with an increasing reward, it shows that the agent is becoming more stable and policy updates are becoming smaller as the agent converges to a better policy.

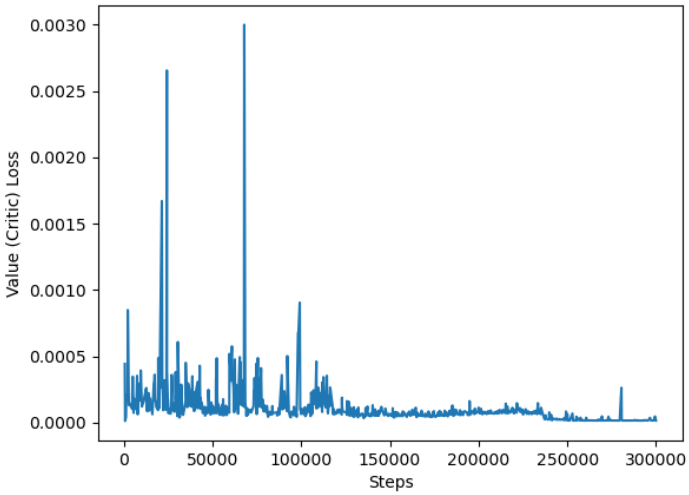


Figure 11: Critic/value loss plotted over time. This indicates how well the value function is approximating the true value of states over time.

As stated earlier, Actor-Critic methods make use of both a policy and value function that are approximated by neural networks. A2C is a policy gradient method primarily but makes use of value function approximation to act as a second opinion when making policy updates. Both graphs depict decreasing loss over time. This means that the network's policy function and value function approximations became more accurate approximations of the real policy and real value functions over time. There is significant variance early on, however, because the optimal policy isn't known yet and so the target that the network is trying to approximate is constantly updating. When the target is non-stationary, it is expected that loss would initially be higher. It is for this reason that these metrics, while helpful in assessing the stability of training, are not as reliable as the average reward over time. The next useful training metric is depicted in Figure 12.

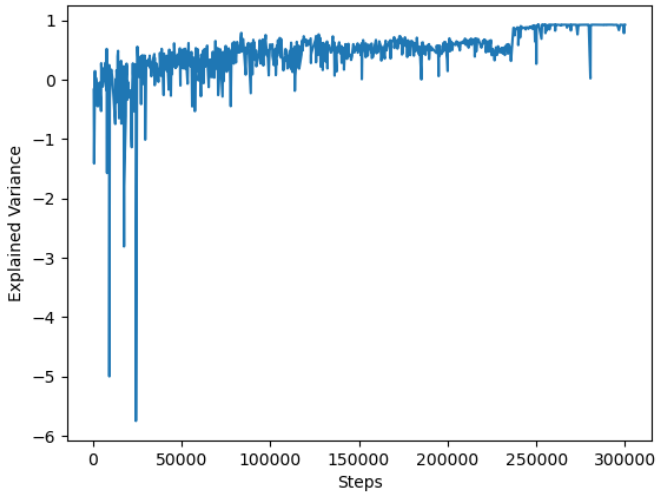


Figure 12: Explained variance plotted over time. This indicates the critic's success in predicting the returns of the policy over time.

Explained variance is a measure of how well the value function, or the critic, is able to predict the actual returns of states. It measures whether or not the value function can explain the variance in the returns, hence the name. It is scaled from 0 to 1, with 0 indicating that the value function is absolutely unable to explain the variance in the returns which indicates that the agent is taking completely random actions. A value of 1 means that the value function can perfectly explain the variance in the returns and indicates that the agent is taking completely deliberate actions. Sometimes the explained variance can be negative. This indicates that the value function is worse than a random prediction and there is something fundamentally incorrect. It can be seen that in this environment the explained variance is initially very negative before finally improving. Towards the end of the training, the explained variance is approaching 1 which is desirable.

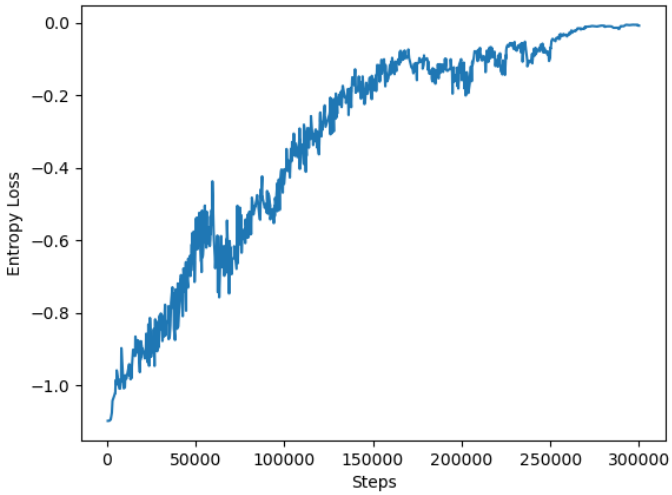


Figure 13: Entropy loss plotted over time. Entropy loss approaches zero over time indicating that there is less need for exploration as the agent approaches an optimal policy.

Finally, we have the entropy loss function depicted in Figure 13. Entropy loss is a term added to the overall loss function calculation of the algorithm that is meant to encourage more exploration and keep the policy from becoming overly deterministic too quickly. The reason that the plot shows the metric starting from the negative and approaching 0 over time is that the entropy loss term is negative and is subtracted from the sum of the other two loss terms. Over time, entropy is reduced to allow the policy to become more deterministic as the agent learns an optimal strategy.

3.5 Slew Path Optimization Results - Reward Type 2

The agent was also trained using the same environment but with a different reward scheme. This time the agent received the same reward type (the normalized RSO count) as before but only if the RSO count increased with each observation. If the agent did not observe more RSOs at the current time step than the previous it would receive a reward of 0. The average episode reward and average episode length are depicted in Figure 14 and 15, respectively. With this reward scheme, the agent may encounter moments where there are no higher RSO count observation windows within the immediate vicinity, meaning it would have to suffer a loss and develop its strategy around this obstacle.

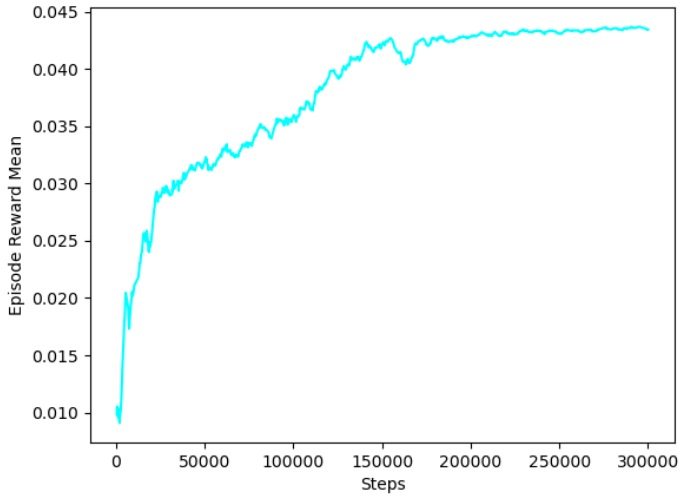


Figure 14: Average episode reward over time with reward scheme 2.

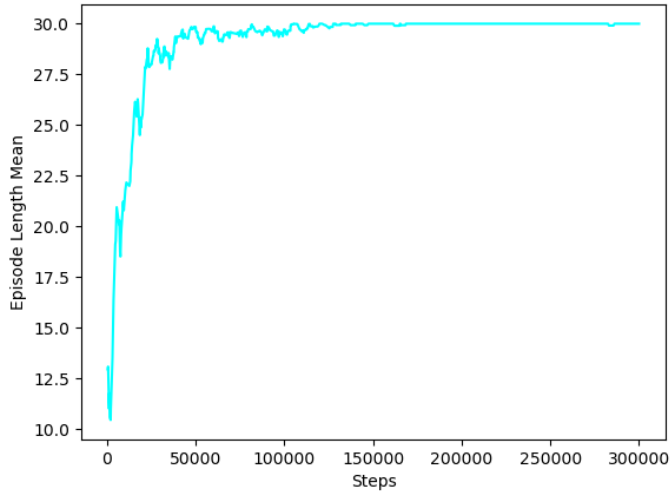


Figure 15: Average episode length over time with reward scheme 2.

Figures 16-19 show the plots of the other performance metrics during the agent's training. One notable difference is that the explained variance grew more linearly this time. With this reward scheme, the agent also took slightly longer to learn the boundaries of the FOR.

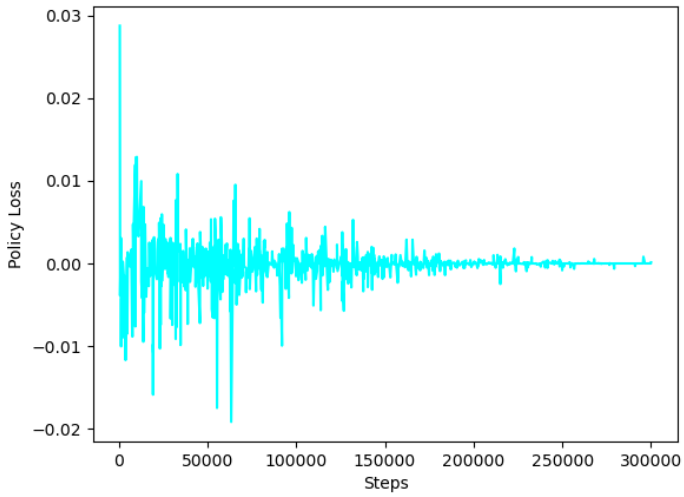


Figure 16: Actor loss over time with reward scheme 2.

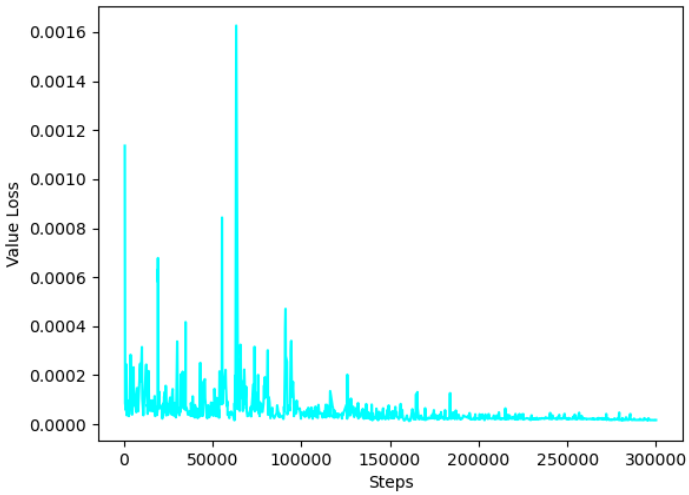


Figure 17: Critic loss over time with reward scheme 2.

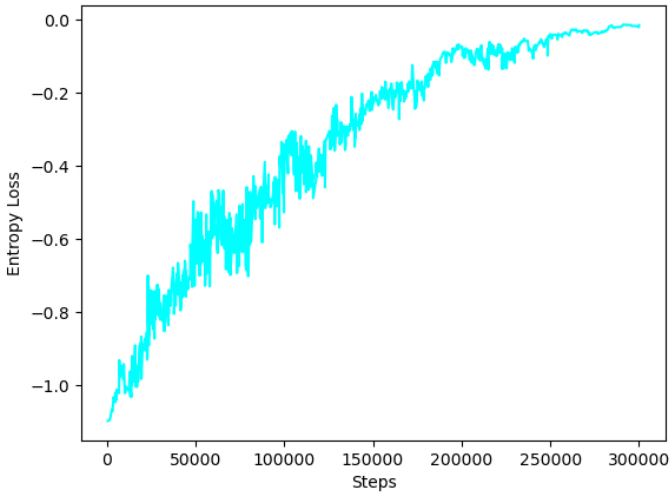


Figure 18: Entropy loss over time with reward scheme 2.

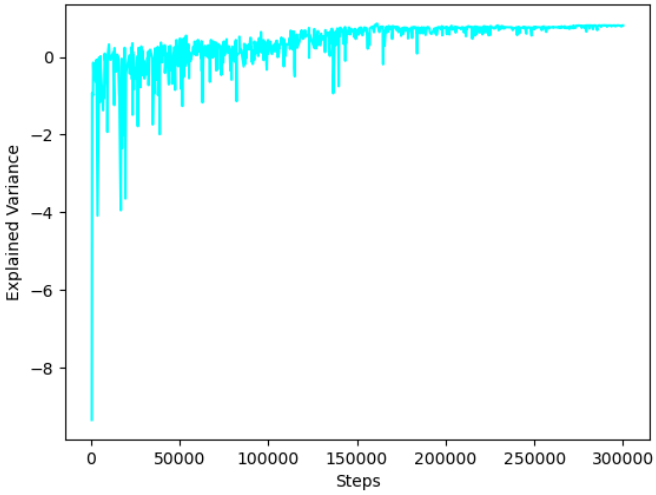


Figure 19: Explained variance over time with reward scheme 2.

Ultimately, the DRL agent successfully optimized this task. This demonstrates that reward schemes are easily customizable in this framework and agents can contend with problems where there may not be immediately rewarding, exploitable actions available at every time step.

3.6 RSO Tracking Optimization

The second SSA scenario explored in this thesis was one in which RSOs would be given tracking priority based on their signal-to-noise ratios (SNR). In SSA, the SNR of an RSO is a measure of the clarity of its emitted signal against background noise. The nature of this signal depends on the sensors used, as this would determine what characteristics are being observed. In the context of optical sensors, SNR generally pertains to how bright and reflective the RSO is with respect to the background of space where light is emitted from many bright sources, such as stars, planets, and galaxies. A higher SNR value means objects are easier to track, and can therefore be tracked more accurately because position and velocity can be measured more accurately. RSOs with higher SNR values can also be tracked at a greater range. This makes SNR a valuable FOM to consider when discussing optical sensor tasking for RSO detection.

SNR in the context of optical sensors has many components. The signal itself is the photon flux (F) received by the detector and multiplied by the exposure time and the quantum efficiency of the sensor. The quantum efficiency (η) quantifies how effective a sensor is at converting incoming photons into electrons. The noise is comprised of several components, as noise can come from many sources when discussing optical sensors. The first of these noise components is associated with the signal itself, and is referred to as the shot noise. Shot noise comes from the inherent inconsistency in the number of photons that can be expected to strike a detector in a given time interval. Then there is the background noise from other illuminated sources, the dark current, which is current flowing through the detector in the absence of any incident photons and is temperature dependent, and the read noise, which is the noise associated with the conversion of collected charges into a digital image. More information can be found in [54].

$$SNR = \frac{\eta F t}{\sqrt{(\eta F + \eta B + D)t + R^2}} \tag{19}$$

In this scenario, the agent controls which RSO is being tracked at any given time. The agent selects an RSO as an action by its catalog ID and tracks it indefinitely unless it discovers that another RSO has higher SNR values at a particular time. For this scenario, a specific SNR threshold of 100,000 was set and the agent would only receive a reward if it was tracking an RSO with an SNR

value above that threshold. This reward was the scaled SNR value on a scale from 0 to 1. The agent would receive a reward of 0 for selecting RSOs that did not meet the SNR threshold. A reward of 0 was chosen for this over a negative reward so as not to discourage the agent from exploring all options. Equation 11 shows the reward function applied in this scenario.

$$r(a) = \begin{cases} \frac{\text{SNR Observed}}{\text{Scaling Value}} & \text{if } SNR > \text{Threshold} \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

This scenario was optimized using two different approaches. First, a DQN was used. In the OpenAI Gym environment, the agent has 6 RSOs to choose from. RSOs are modeled as box-wing satellites using a custom facet model based on the physical and material characteristics of Intelsat 10-02. The agent selects an RSO by its catalog number from a list as an action and this catalog number is used as an input to the MATLAB script running the tracking simulation. The simulation itself is of a 42-minute period of observation with 1-minute time steps. The targets and length of observation were chosen based on whether those targets met the set SNR threshold at any time during the observation period. A 1-minute time step was chosen so as to capture changes in SNR without having to simulate every second of the scenario, as this would take up far too much training and simulation time.

3.7 Deep Q Network

The DQN algorithm has the same default architecture in SB3 as the A2C algorithm. It does, however, have a variety of different hyperparameters unique to the value-based optimization approach. As with A2C and all other algorithms, the learning rate is adjustable. By default, this value is 0.0001 for DQN and this was maintained for the purposes of this research. Beyond that, the buffer size is an adjustable hyperparameter as well. Given that this is a DQN, it uses a value-based policy optimization approach, utilizing an experience replay buffer that allows the agent to call upon past experiences to improve the learning process. By default, this buffer can store 1,000,000 experiences. This is equivalent to 1,000,000 time steps. The replay buffer is a circular buffer, meaning that old experiences will gradually be discarded to make room for new experiences when the buffer becomes full. In this scenario, the buffer size was set to 100,000 because the agent was trained for

less than 140,000 time steps (although the episode length was set to 420,000) and, by restricting the buffer size, the agent would be forced to sample from newer, more relevant experiences.

The next hyperparameter of note is the "learning starts" hyperparameter. This defines how many experiences should be collected before the agent can start learning from them. If learning begins too early in DQN, the agent will have insufficient experience to draw on and that would likely cause bias. The default value is 100, and this was left unchanged since this scenario has a short episode length and is not very complex. Another related hyperparameter is the batch size. The batch size value determines the number of experiences that will be sampled throughout the training process. This value is 32 by default.

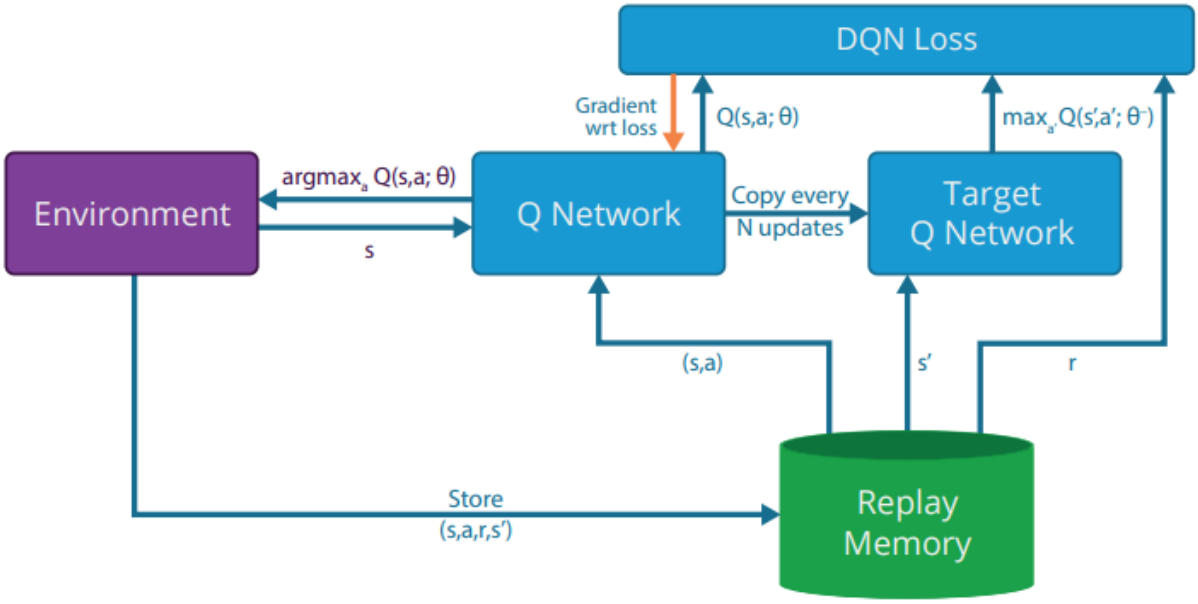


Figure 20: DQN architecture diagram [48].

Additionally, DQN has the "tau" hyperparameter. Tau controls how strongly the target policy is updated to match the behavior policy when target policy updates take place. This means that the user can control how gradually the target policy is updated to resemble the behavior policy. By default, this value is 1. A value of 1 means that the target policy is updated to completely match the behavior policy each time the target policy is updated. A value lower than 1 means that the target policy will not match the behavior policy after a target policy update, which is known as a soft update. The soft update formula is expressed in Equation 21, where θ_{target} is the target policy and θ is the current behavior policy.

$$\theta_{target} = \tau\theta + (1 - \tau)\theta_{target} \tag{21}$$

Then there is gamma, the discount factor. By default, this is 0.99 and this value is commonly used across all DRL algorithms, though it can be adjusted. The next important hyperparameter is training frequency. Training frequency determines how many steps are taken within the environment before the policy is updated. In most applications of value-based methods, the policy is typically

updated after every step. This value is 4 by default but was set to 1 when this scenario was run so that updates would occur with every step. Batch updates can be used, however, the Q values are still updated after every step. The difference with larger batch sizes is that Q value updates are applied to the policy less frequently which helps to break the correlation between past and newer experiences and allows the agent to explore more trajectories.

There are other options, such as the gradient step value, which determines the frequency of network updates relative to the number of steps taken within the environment. There are also a few optional hyperparameters pertaining to the configuration of the replay buffer. None of these were changed for this scenario, however. The next hyperparameter that was changed was the target network update interval. This determines how many steps occur before the target network is updated with the strategies learned by the behavior policy. After this, the exploration fraction can be set. The exploration fraction determines what percentage of the total training length is spent collecting experiences through ϵ-greedy behavior. There are no specific best practices in determining the exploration fraction. By default, it is set to 10% of the training length. However, it was set to 25% of the total training length when training the agent on the tracking scenario to ensure a diverse collection of experiences. The starting value of ϵ can be set by the user as well. By default, ϵ is set to 1 so that the agent begins by behaving completely randomly. It gradually decreases over time and stabilizes at 0.05 so that the agent is only taking a random action 5% of the time, though this can be changed as well. Table 2 shows the values of the DQN hyperparameters.

Hyperparameter	Setting
policy	“MlpPolicy”
env	env
learning_rate	0.0001
buffer_size	100000
learning_starts	100
batch_size	32
tau	1.0
gamma	0.99
train_freq	1
gradient_steps	1
replay_buffer_class	None
replay_buffer_kwargs	None
optimize_memory_usage	False
target_update_interval	1000
exploration_fraction	0.25
exploration_initial_eps	1.0
exploration_final_eps	0.05
max_grad_norm	10
stats_window_size	100
tensorboard_log	“file\path”
policy_kwargs	None
verbose	1
seed	None
device	“auto”
_init_setup_model	True

Table 2: DQN hyperparameter configuration for tracking optimization scenario.

3.8 RSO Tracking Optimization Results - Deep Q Network

The results from the DQN approach demonstrate the difference in training approach. Under the value-based approach, the agent spent the first 50,000 time steps collecting experiences from a set of randomly initialized parameters. As is shown in Figure 21, the first 50,000 time steps show no demonstrable improvement in the average reward over time. At this stage, the behavior of the agent was almost entirely random due to the size of the ϵ exploration value. At 50,000 steps, the agent began to exploit its collected experiences as the ascent in the average episode reward became almost vertical. After this point, there was a more gradual increase as ϵ decayed and the network continued improving its Q-value estimates. The reward began to plateau past 100,000 time steps as the agent began approaching the optimal policy. However, there is a noticeable decline after

130,000 time steps. The reasons for this downward turn are not entirely clear, although it may be a consequence of restricting the replay buffer size by an order of magnitude. At that stage in training, the replay buffer had already lost a large number of older experiences that had been overwritten by newer ones. The initial thinking behind restricting the replay buffer was that overwriting older experiences with newer ones over time would direct the agent toward more valuable actions. It is difficult to know how the manipulation of algorithm hyperparameters will precisely affect the training outcome of an agent, as there are no established best practices for DRL training. Different environments may benefit from different hyperparameter values, which is why those who study DRL engage in hyperparameter tuning, whereby researchers try many different combinations of hyperparameters. There is a lot of trial and error in this approach, although researchers will often try to draw on their intuition and understanding of the underlying mathematical framework of DRL.

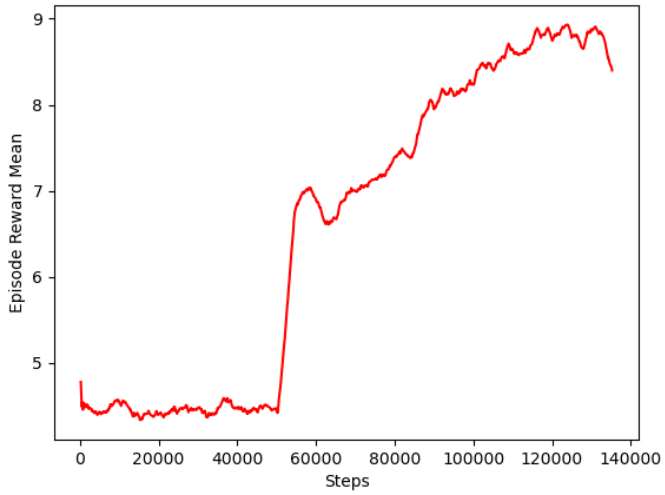


Figure 21: Average episode reward over time for DQN.

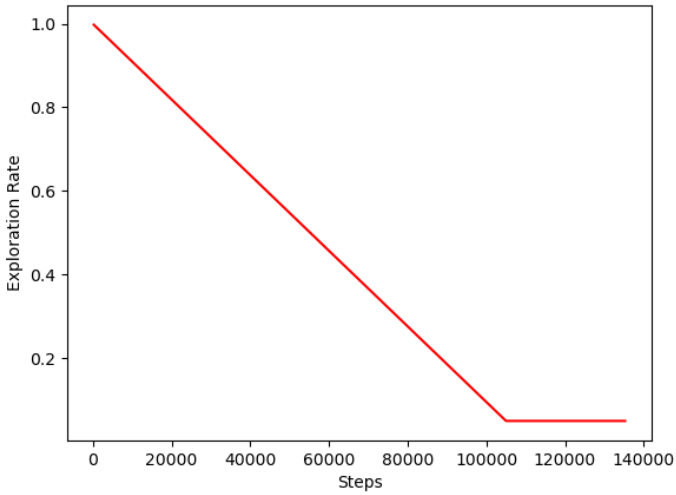


Figure 22: Epsilon decay over time for DQN. The epsilon value determines the frequency with which random actions are chosen for the purposes of exploration. This value decays over time to approach a more deterministic policy.

Figure 22 shows the decay of the exploration rate, ϵ, over time. When training begins, the value is at 100%, meaning that the agent's behavior is entirely random. It decayed to a base rate of 5% exploration so that most of the agent's actions would be deliberate exploitation with occasional random actions included. Despite the low probability of exploration after ϵ decays to its base value, the agent still actively samples the replay buffer and updates the Q values based on these sampled experiences.

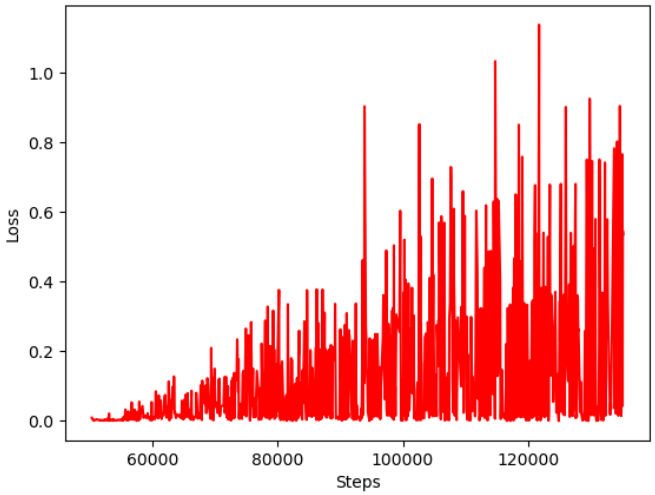


Figure 23: Network loss over time for DQN. This shows how well the Q-network is approximating optimal Q-values over time.

Figure 23 shows the DQN loss over time. This graph exemplifies why loss in the DRL context is not always a reliable indicator of algorithm performance. The graph demonstrates a growth in loss over time despite the agent converging to an optimal policy, because the network's target policy is non-stationary and evolves as the agent collects more information about the environment.

3.9 Proximal Policy Optimization

Proximal Policy Optimization can be described as an extension of A2C. Both algorithms take the policy gradient approach to policy updates and both make use of an actor and critic network to make advantage estimates, which is used in the gradient estimate calculation. As previously mentioned, the gradient estimator function of the vanilla REINFORCE policy update approach takes the form of Equation 15. In A2C and PPO, the expected return is replaced with the advantage function, which measures how much better or worse an action is compared to the expected outcome of a state. With this change, Equation 15 is replaced with Equation 24.

$$\nabla J(\theta_t) = \mathbb{E}_\pi \left[A_t \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \tag{22}$$

PPO adds some additional features that improve upon the A2C platform. These changes make

training more stable and sample-efficient. One improvement is the introduction of mini-batches. Whereas A2C would perform gradient updates after each batch, PPO can perform multiple updates per batch through the use of mini-batches. The policy will also perform multiple epochs of updates on each mini-batch. This leads to better sample efficiency and higher training stability by making policy updates more gradual. Batch data is stored in a rollout buffer, which functions similarly to a replay buffer in DQN. The difference, however, is that the rollout buffer only stores data that is actively contributing to a current policy update. After the update is complete, this data is discarded. Past experiences are not stored and cannot be used for learning like in DQN due to the explicit nature of how policy updates are performed in policy gradient methods.

PPO introduces another method of controlling the size of policy updates as well. This comes in the form of a clipping mechanism in the surrogate objective function, another feature introduced in PPO, that clips the probability ratio of the new policy and the old policy so that it is within set boundaries [49]. This keeps policy updates from being too large and destabilizing training. The surrogate objective function itself, as the name implies, acts as a surrogate for the modified REINFORCE objective function $J(\theta)$ found in A2C. This surrogate function is represented in Equation 23.

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t, clip \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \right] \tag{23}$$

The clip term clips the probability ratio within the bound set by 1 - ϵ and 1 + ϵ, where ϵ is the clipping parameter that limits how far the probability ratio of the new and old policy is able to deviate from 1 [49]. $\hat{A}_t$ represents the advantage function. The hat symbol is used to denote that the advantage function is an approximation, rather than the true advantage function, as the term Q(s, a) is not calculated directly in policy gradient algorithms. It is perfectly acceptable to write the advantage function without the hat symbol, however.

PPO also sometimes incorporates a Kullback-Leibler (KL) divergence penalty term into the objective function. Kullback-Leibler divergence, named after mathematicians Soloman Kullback and Richard Leibler, refers to the difference between two probability distributions taken over the

same variable. In the case of PPO, it quantifies the divergence of a new policy from an old one. In the SB3 PPO algorithm, the KL divergence term is not incorporated into the objective function. The SB3 algorithm only relies on the clipped surrogate objective function, although the KL divergence is a metric that is monitored during training.

In SB3, the PPO algorithm has many of the same available hyperparameters as A2C and some new additions that account for PPO's improvements over A2C. The mini-batch size can now be specified, as well as the number of epochs per batch. The clip range, or clipping parameter, can be adjusted for both the policy and value functions in PPO. Users can also choose whether or not to set a limit to KL divergence between policy updates.

For the tracking scenario, the number of steps per update was set to 42, which is the length of the scenario in minute time steps. Batch size was set to 14, meaning there would be 3 mini-batch updates per episode. Table 3 shows the PPO hyperparameter values.

Hyperparameter	Setting
policy	“MlpPolicy”
env	env
learning_rate	0.0003
n_steps	42
batch_size	14
gamma	0.99
gae_lambda	0.95
clip_range	0.2
clip_range_vf	None
normalize_advantage	True
ent_coef	0.0
vf_coef	0.5
max_grad_norm	0.5
use_sde	False
sde_sample_freq	-1
rollout_buffer_class	None
rollout_buffer_kwargs	None
target_kl	None
stats_window_size	100
tensorboard_log	“file\path”
policy_kwargs	None
verbose	1
seed	None
device	“auto”
_init_setup_model	True

Table 3: PPO hyperparameter configuration for tracking optimization scenario.

3.10 RSO Tracking Optimization Results - Proximal Policy Optimization

The results presented in this subsection demonstrate a clear advantage in the DQN approach to optimizing the RSO tracking scenario over PPO. While this is not a conclusive determination, it is a fact that DQN is particularly adept at determining policies when environmental states are and actions are discrete. Due to the size of the time steps in this environment, the differences in SNR values from one state to the next are quite large. This means that there are very well-defined state transitions and distinct state-action pairs. PPO is an algorithm designed for environments where granular control is required, and where long-term strategies and broader trajectories need to be considered. Essentially, the object tracking scenario in this form is too simplistic to be well optimized by PPO's particular approach to optimization. PPO, along with policy gradient methods

in general, optimizes policy across entire trajectories because policy gradient methods focus on explictly optimizing the entire policy. DQN implicitly develops optimal policies by estimating the value of discrete state-action pairs at each time step. At the end of this subsection, the policies developed by DQN and PPO for the tracking scenario will be compared with each other, and with the global optimal policy that was acquired manually by running the simulation in MATLAB.

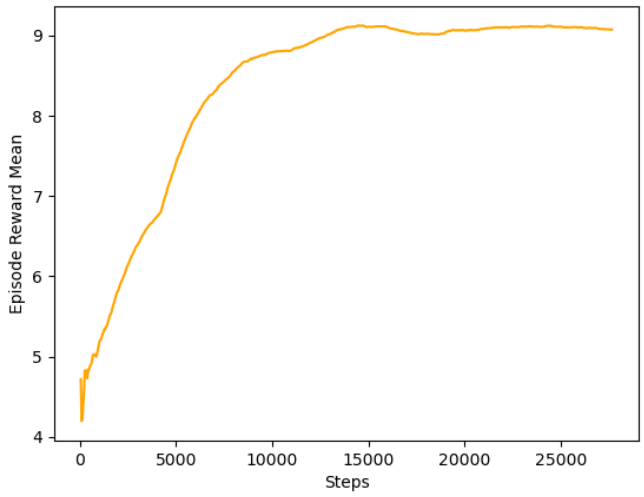


Figure 24: Average episode reward over time for PPO.

From Figure 24, it is clear that the PPO-based agent was able to achieve a peak average episodic reward close to that of the DQN-based agent at its peak in far less time. This is not unexpected, as the scenario is simple enough that it could be optimized manually, and the DQN agent spends a considerable amount of time collecting experiences instead of exploiting known good actions.

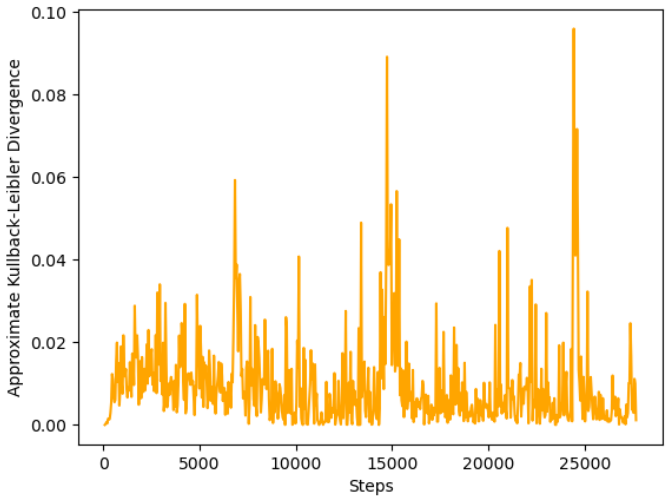


Figure 25: Approximate Kullback-Leibler divergence over time for PPO. This shows how far new policy updates diverge from old policies over time.

Figure 25 shows the approximate Kullback-Leibler divergence between new policies over time compared to the previous policies. This particular scenario was only trained for a short period of time, so it may not have been a long enough duration to determine a trend in KL divergence. However, apart from some large spikes, it does appear to be trending downward, which is ideal. Ideally, as the agent approaches an optimal policy, the gap between the older and newer policies should diminish.

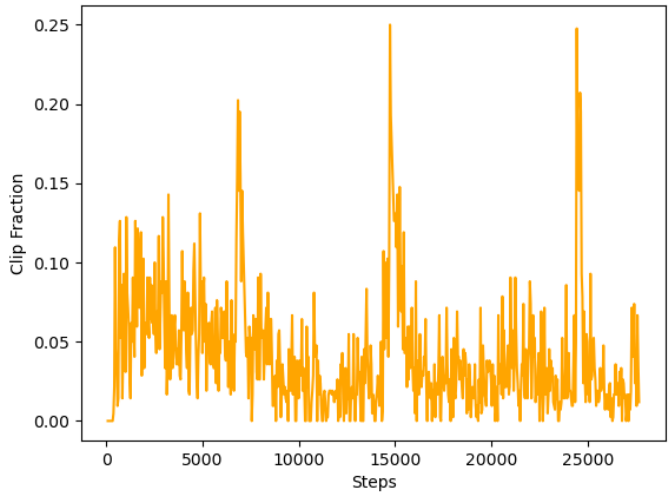


Figure 26: Clip fraction over time for PPO. This gives the proportion of policy updates that exceed the clip range. Large spikes in the clip fraction signify aggressive policy changes, so a downward trend signifies increasing stability in policy updates.

Figure 26 shows the clip fraction over time. The clip fraction signifies the proportion of policy updates at a given time that are clipped, meaning they exceed the clipping range of the algorithm. Overall, the clip fraction should trend downward or remain stable under 0.10. This signifies training stability. With the exception of a couple of large spikes, the clip fraction is evidently trending downward and stays below 0.10.

Other metrics can also be seen converging or seemingly converging to desired values over time in Figures 27 through 31.

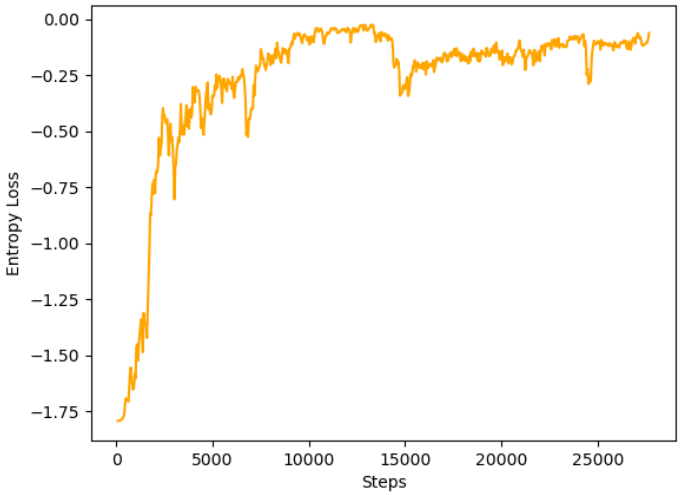


Figure 27: Entropy loss over time.

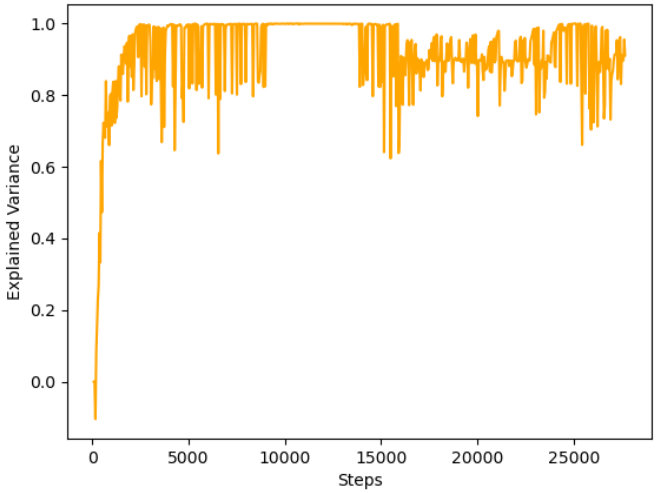


Figure 28: Explained variance over time.

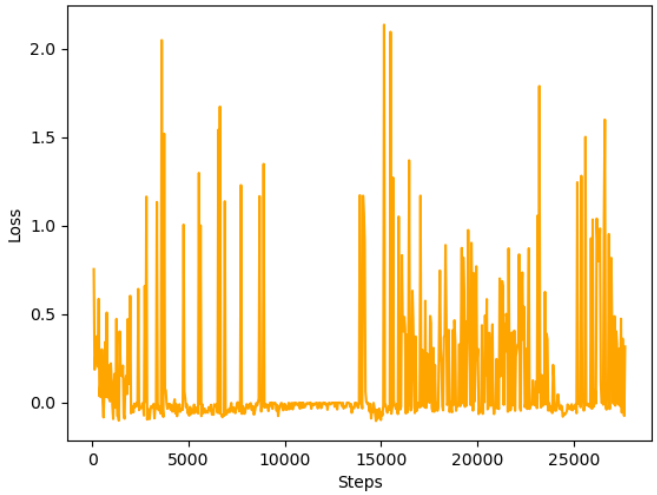


Figure 29: Overall loss over time.

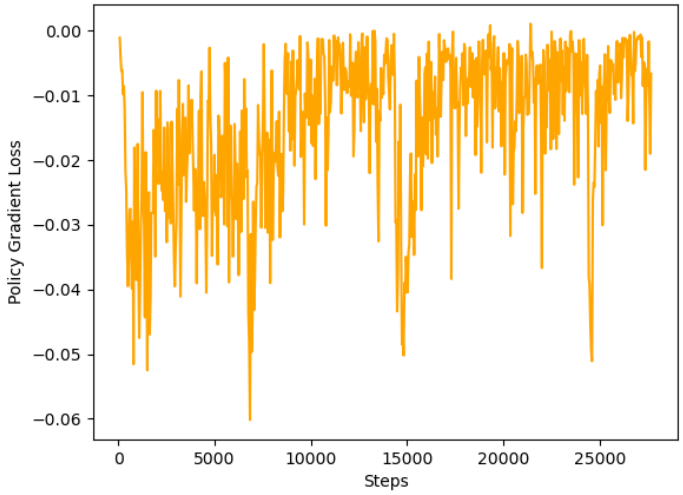


Figure 30: Policy gradient loss over time.

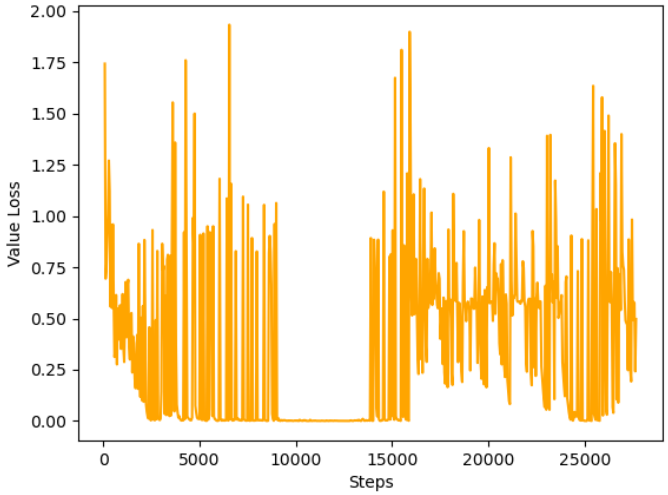


Figure 31: Value loss over time.

While these metrics indicate that the agents are learning optimal policies, it is worth comparing the policies themselves to see how close they are to the global optimal policy. Fortunately, due to the simplicity of the object-tracking scenario, this was possible. The following diagrams display the policies with the highest average rewards before training was terminated. They show the RSOs chosen for tracking by their catalog ID in the order that they were chosen from the beginning of the scenario to the end.

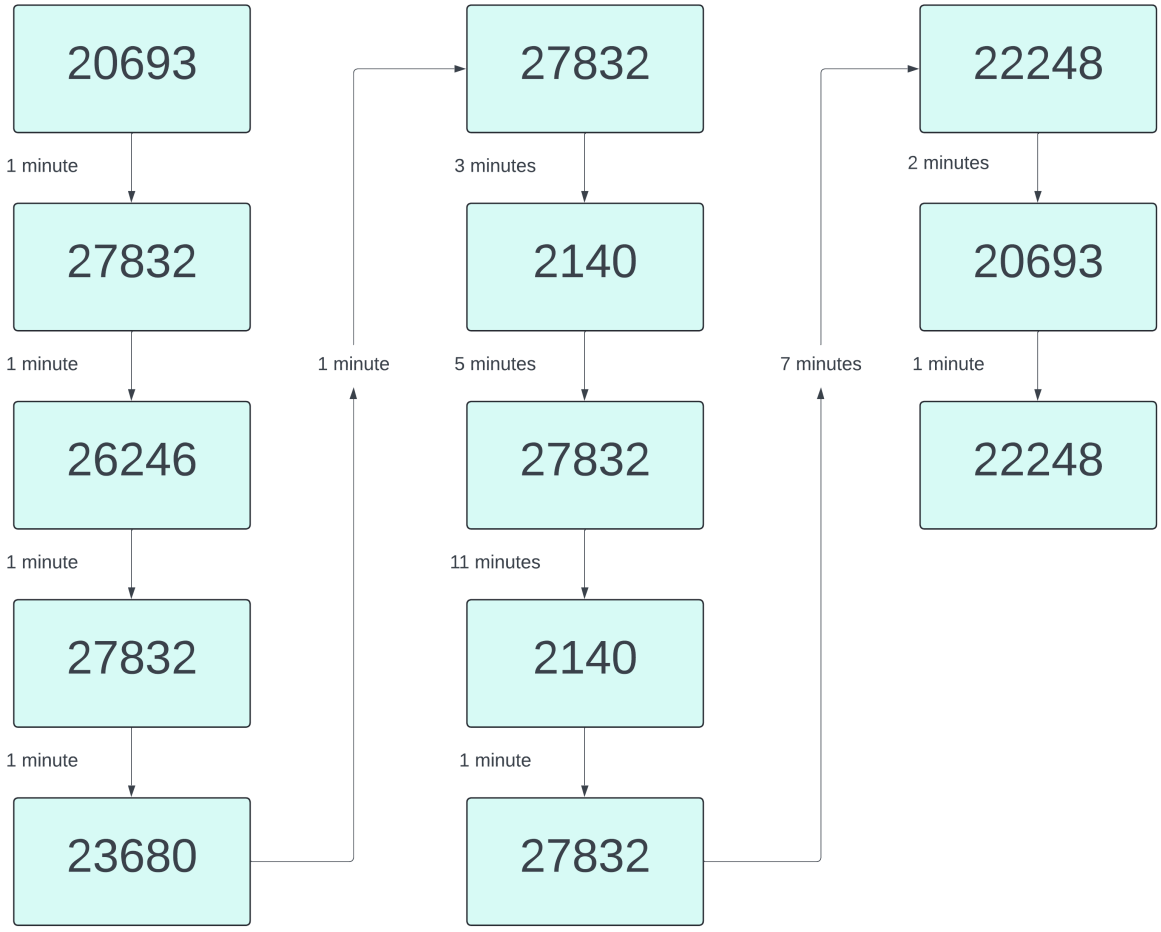


Figure 32: This figure shows the global optimal policy as determined manually through several runs of the MATLAB simulation, tracking each object for the entire duration of the simulation. There are 6 objects in total to track and each box represents an RSO identified by its unique catalog number. The order of each box indicates the order each object should be tracked in. The labels indicate the optimal length of time each object should be tracked.

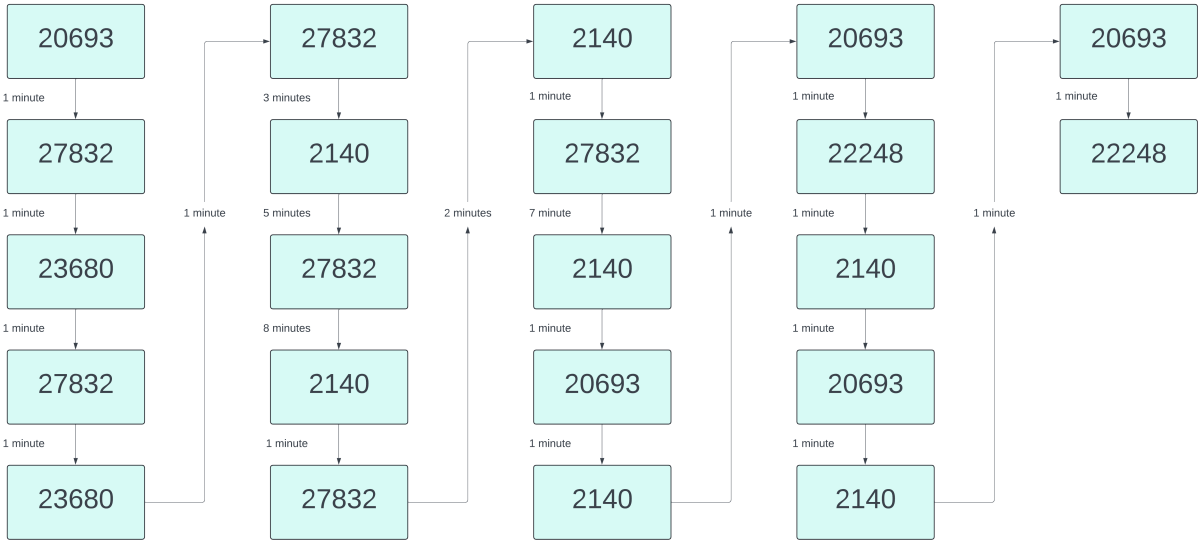


Figure 33: DQN policy with the highest average reward for the object tracking scenario.

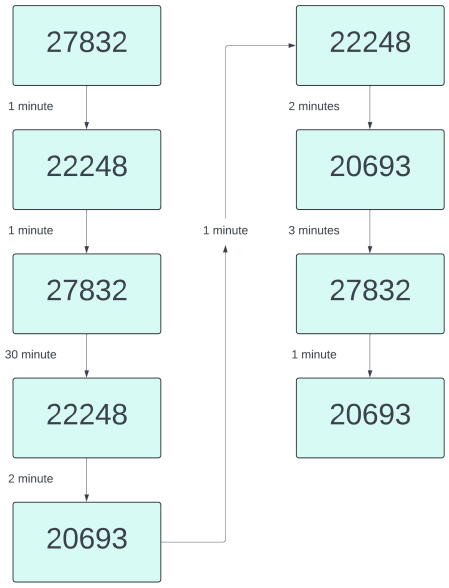


Figure 34: PPO policy with the highest average reward for the object tracking scenario.

Figures 32 through 34 show the order in which RSOs were chosen for observation. When comparing the order on the diagrams, it is clear that the DQN policy is closer to the globally optimal policy. However, the end of the DQN policy deviates significantly from the optimal policy. This is likely due to a bias developed towards the earlier, more rewarding RSO choices. Further training could resolve this problem.

Conversely, the PPO algorithm more correctly captures the optimal RSO tracking policy at the end of the sequence but takes a broad approach to the first approximately three-quarters of the sequence and misses out on more optimal RSO choices. Overall, the DQN agent tracked the most optimal RSO for 32 minutes out of 42. The PPO agent tracked the most optimal RSO for 27 of the 42 minutes.

4 Final Remarks

4.1 Summary

To summarize, this portion of the thesis explored the application of deep reinforcement learning algorithms in the context of optimizing space situational awareness scenarios. Two SSA sensor tasking scenarios were explored in order to demonstrate the potential of DRL algorithms as alternatives and, in some areas, an improvement over traditional methods in sensor tasking optimization.

The first of these scenarios explored the benefit of using a DRL agent to adjust a satellite's optical sensor to navigate a region of space so as to maximize unique RSO detections. Through the use of the A2C algorithm, the agent's approach to this scenario was iteratively updated over time toward an increasingly optimal solution. Within 10,000 rollouts of the scenario, the agent began to successfully converge to an optimal solution despite a very large number of trajectories. It was able to learn the boundaries of the observation area, and which regions of the FOR to observe at a given time based solely on simple reward feedback from the environment.

While the global optimal solution to the problem is not known due to the vast number or possible trajectories, the training metrics demonstrate the self-correcting and intelligent decision-making capabilities of DRL algorithms. Unlike more traditional optimization techniques, which start with a large pool of candidate solutions and discard those that are ranked lower according to fitness functions, DRL agents start with a single policy and iteratively improve upon it by updating internal network parameters. In other words, DRL agents are capable of "learning," while more traditional optimization algorithms do not.

Beyond this, DRL algorithms offer advantages that traditional optimization techniques are incapable of replicating, such as the ability to take image data as input. DRL algorithms can also be run on GPU devices, allowing them to take advantage of GPU parallel processing. Furthermore, DRL offers multi-agent training that can rival swarm optimization's similar multi-agent approach.

Despite the optimal solution to the first scenario being difficult to verify empirically, the optimal solution to the object tracking scenario is known. The object tracking scenario, where RSOs were

given tracking priority based on their SNR, was an attempt to explore a scenario similar to those explored in [14, 15, 17]. These scenarios also involved a DRL agent having to prioritize which RSOs to track based on certain criteria; namely, in their case, the size of the error in RSO position estimates.

Two different agents based on two different algorithms and approaches were trained on the tracking scenario. These algorithms were DQN and PPO; a value-based approach versus a policy-based approach to optimization. The DQN and PPO agents were trained for around 140,000 steps (around 3300 episodes) and around 30,000 (around 700 episodes) steps, respectively. Agents were able to choose which RSOs to view at each time step. The DQN agent proved to be more capable of achieving a near-optimal policy than the PPO agent in this particular scenario. While the reason is not entirely clear, it is likely related to the differences in how these two optimization approaches estimate optimal behavior. The DQN algorithm's implicit approach to policy optimization focuses on the value of taking specific actions in environments with discrete states and actions.

The main goal of this research was to demonstrate the applicability of DRL algorithms to different SSA optimization problems beyond those previously explored. There are many potential applications of DRL in SSA, even beyond sensor tasking. DRL algorithms could potentially be used to teach agents collision avoidance and autonomous maneuvering, as well as orbit maintenance. The research completed in this part of the thesis represents a modest exploration of DRL in two SSA toy problems and only scratches the surface of the depths to which this particular optimization method can expand our SSA capabilities, particularly in autonomous sensor tasking. It is clear that DRL offers a considerable degree of flexibility and can seemingly be adapted to any SSA problems that involve sequential decision-making.

DRL may not necessarily replace other forms of optimization algorithms currently used in SSA, but it can approach sensor tasking using a unique, trial-and-error strategy based on simple reward feedback that leads to self-correction and "learning". With the correct application, DRL agents can learn to generalize beyond their training environments by identifying patterns in a variety of input data. This is in stark contrast to other, heuristic-based optimization algorithms that can only rank entire sensor tasking schedules according to user-defined fitness guidelines.

4.2 Future Work

Deep reinforcement learning offers a powerful framework for solving optimization problems. With DRL, there are many algorithms to choose from based on different approaches to policy optimization, and there are many ways in which these algorithms can be configured prior to training an agent.

The research discussed in this thesis, for the most part, makes use of the default SB3 A2C, PPO, and DQN algorithms. Some hyperparameters were changed prior to training based on notions of how they may impact training. However, as stated earlier, DRL algorithms are an ongoing subject of research and there are no concrete established best practices like the ones found in supervised learning, for instance. Hyperparameter tuning and trying different network architectures exceeded the scope of this research, as the effects of these require a significant amount of time to study.

The hardware the simulations and DRL training sessions were run on was an i7-9750H CPU. DRL training periods are as long as the user defines. In the previous sections discussing training scenarios, the longest training period discussed of 300,000 time steps was equivalent to nearly 4 days of real time. The next largest training period discussed in the previous sections was 135,240 time steps in length, which took over 2 days to complete in real time. Finally, one particular DRL model discussed in the following sections was trained on a simulation for 27,678 time steps, which comes out to more than 14 hours of real time. Due to the length of time required for training DRL models on these simulations, it left little time for testing the effects of different hyperparameters or even different network architectures. A considerable amount of time was spent ensuring the simulations were set up correctly, and that the OpenAI Gym environmental wrappers were functioning as intended. Sometimes an issue wouldn't become obvious until training had already commenced and the agent was not learning from the environment. Given that a DRL agent requires time to explore the environment initially, it is not immediately obvious if an agent's poor performance is due to a technical issue, or a lack of sufficient experiences. A great deal of time was also spent determining what kinds of scenarios would best showcase DRL performance in SSA challenges. However, if this research were to be extended, these would be included within the scope of the research.

Additionally, due to time constraints, additional environmental and physical limitations, such as

power availability or structural constrictions that might limit sensor range of motion for example, were not incorporated into simulations. With more time, complex constraints and even actions, such as attitude maneuvers, could have been factored into these optimization scenarios. Additonal figures of merit such as relative distance could have been incorporated into these optimization scenarios as well. This would allow for the creation of more complex reward functions that factor in the resource cost of performing certain actions and would force the DRL agents to balance long-term goals with the immediate physical costs of their behavior. For instance, there could be penalties based on how far the agent (sensor) would have to slew from one object to another, or the extent of attitude maneuvers that would need to be performed in order to continue observing particular RSOs. This would have made the scenarios significantly more realistic simulations of real sensor tasking problems. If this research were extended, these additional considerations would be actively explored.

Part 2: Object Detection

This portion of the thesis focuses on RSO detection from simulated and real satellite images using R-CNNs. Expanding object detection capabilities in space is crucial for SSA and the leveraging of AI for autonomous object detection has been an active area of research for some time. Convolutional neural networks can be trained to be highly robust and accurate in detecting patterns even in challenging visual conditions. This is crucial in the SSA context where image quality is affected by many bright sources and visual artifacts resulting from hardware itself.

Chapter 5 gives offers an overview of object detection algorithms, beginning with those that do not fall under the machine learning category. Next, chapter 5 discusses machine learning detection algorithms, along with a brief overview of how they learn to identify and classify objects. This leads into a description of each iteration of R-CNNs leading up to the main subject of this section, which demonstrates RSO detection via Faster R-CNN.

Chapter 6 provides information on the datasets used for training the Faster R-CNN implementation, as well as the methodology and results of training. The methodology includes the configuration of the CNN backbone of this Faster R-CNN implementation, as well as the chosen settings for the region proposal network (RPN) and region of interest (RoI) pooler. The "results" sections include the results of training on the simulated image data, as well as the real image data.

Chapter 7 includes a summary of the work showcased in Chapter 6, as well as conclusions drawn from the results. The results demonstrate the capability of Faster R-CNN as a robust algorithm for RSO detection in real-time with a high degree of accuracy. The promising results of this study suggest that Faster R-CNN should be explored further as competitive object detection model for SSA. This chapter concludes with a section on future work if the scope of this thesis were to be extended.

5 Object Detection in SSA

RSOs are often identified from images taken by both ground and space-based observatories with optical sensors. RSOs can often, though not always, be identified as streaks or small blobs moving across sets of these images against static star backgrounds. While these moving objects can be identified by a human being, it is vastly more efficient and effective to employ computer algorithms to perform these tasks autonomously, with advanced computer vision techniques often considered for such applications. Computer vision algorithms can assist in the automated identification of objects from both ground and space-based platforms with a high degree of accuracy. Because of this, running these algorithms on space-based hardware in real-time is desirable and can potentially expand our capabilities to detect objects in LEO, since weather conditions and geographical location affect the visibility of these objects from ground-based observatories.When discussing "real-time" detection, this refers to the ability for an object detection algorithm to process images at a rate equal to, or faster than, an optical sensor's frame rate. This ensures that no objects detected by the sensor are missed by the algorithm.

Different computer vision algorithms are currently used for handling SSA tasks; in particular, locating RSOs in satellite images and attempting to determine what type of object they are. To date, there have been limited attempts to apply the most modern and sophisticated machine learning object detection frameworks to low-resolution, real satellite images containing point-source objects. This is not particularly surprising, given that these algorithms are mostly intended to process images of everyday objects for tasks such as autonomous driving. The point source objects found in many optical sensor images are very small and have little in the way of distinguishing characteristics, which is likely why few attempts have been made to train CNNs on this type of image data; aside from the availability of the data itself, that is. However, some of these computer vision algorithms based on machine learning principles and neural network architecture can potentially have a significant impact on RSO detection and classification problems.

5.1 Traditional Object Detection Algorithms

Some traditional computer vision and image manipulation techniques that have been used in SSA include edge detection and pixel magnitude thresholding. Edge detection algorithms like the Canny edge detector make use of thresholding and other image manipulation techniques and have been used in SSA for streak detection [31], for example. Other algorithms include an assortment of frame differencing algorithms. These operate under the simple premise of subtracting images from one another to determine which objects are in motion, as static objects will be largely eliminated if their position is fixed. Another approach used in object detection applications in SSA is optical flow [32]. Optical flow algorithms estimate the velocity of objects in images based on their motion through frames.

The problem with Canny edge detection, or frame difference algorithms, is that both approaches require fixed thresholds to be set by users. This means that if poor thresholds are chosen, important data can be easily missed by the algorithm. There is also a limit to the level of image complexity that these algorithms can handle. Optical flow also suffers from various drawbacks. Optical flow methods perform worse under changing illumination conditions, for example. This is a significant drawback in space object detection, where illumination conditions are far from constant. These kinds of drawbacks can be largely overcome with machine learning.

5.2 Examples of Machine Learning Object Detection Algorithms

There are a number of modern deep learning algorithms that are used for everyday object detection and classification, some of which are employed by companies to solve autonomous driving, healthcare, and surveillance problems. YOLO is a very popular single-shot detection algorithm that performs a single pass on an image and divides the image into a grid of cells, performing bounding box and confidence score predictions for objects whose center coordinates fall within a cell. YOLO is, by default, a large and deep neural network architecture; although there are smaller versions. Being a generally large, single-shot detector means that small object detection could be an issue. EfficientDet [33] is another modern object detection network that combines the EfficientNet [34] network backbone architecture, which is meant to optimize accuracy and computational cost, with

a unique feature pyramid network (FPN). There is also CenterNet [35], which uses the coordinates of object centers to predict object dimensions. This eliminates the need for multiple bounding box proposals. Beyond these, R-CNNs are the main rivals to YOLO algorithms and use a unique network architecture for object detection that involves dividing images into regions of interest that are processed individually. While not quite as fast as YOLO due to their two-stage image processing approach, these algorithms are more than fast enough for real-time object detection and benefit from the usual advantages of two-stage algorithms, such as higher accuracy, generally better ability to detect small objects, robustness in the face of image complexity, and network modularity which is valuable when network size constraints are a necessary consideration.

5.3 Literature Review

Machine learning algorithms have been applied in SSA contexts for a while. Various types of CNN architectures and novel algorithms have been used in SSA applications such as object classification and detection in recent years. For example, the authors of [36] use end-to-end fully convolutional networks for small space debris detection in low-resolution images. They test their network's robustness under different noise and glare conditions, demonstrating a significant trade-off between network precision and recall. The authors of [37] use a modified U-Net CNN to identify object "tracklets" from telescope survey images and use these to determine the object centroid locations in post-processing. In [38] the authors explore the use of simple CNNs for the classification of RSOs and visual artifacts for a set of satellite images taken from the Sapphire mission, with the eventual goal of implementing the algorithm onboard a spacecraft using an off-the-shelf, commercial GPU. They test a fully connected network configuration, as well as two different CNN configurations on images with different levels of added Gaussian noise, demonstrating more robustness in their second, deeper CNN model. The authors of [39] and [40] explore the applications of EfficientDet and YOLO on simulation images of satellites and debris. However, these are highly resolved simulation images and not real satellite images such as those used in the research discussed in this thesis.

An ideal detection algorithm for SSA would be one that can autonomously process images in real-time, can both localize and classify small objects like space debris and be suitable for spacecraft hardware implementation, preferably on a field-programmable gate array (FPGA) device. Based

on the results explored in this thesis, Faster R-CNN appears to be a strong candidate to meet these criteria. The content of this thesis related to object detection, along with [41], represents an initial foray into the exploration of Faster R-CNN for real-time RSO detection and classification with real satellite images.

The RSO detection algorithm presented in [41] identified RSOs in rolling sets of FAI images and persistently matched these detections across image sequences. The algorithm first pre-processed the image set, as described in Section 2.3 of the paper, before using a Faster R-CNN to detect any RSOs present within the images. The algorithm then used a Hough Transform algorithm to match RSO detections between images to the same parent object. The R-CNN detection algorithm had a 73.6% true positive rate and a 32.1% recall rate on a small of 332 labeled images. While these results were not unreasonable, they left significant room for improvement.

5.4 Convolutional Neural Networks

In all supervised learning problems, an algorithm will attempt to find the function that maps an input to an expected output. Neural networks are universal function approximators that attempt to map input data to an expected output. Convolutional neural networks take in data, which is generally image data, and attempt to estimate the function that maps parts of an image to assigned labels and a set of coordinates through the use of convolution operations. This is how object classification and localization are performed.

Image pixel data is represented as a multi-dimensional tensor with height, width, and color channels as the dimensions. This image data is processed by filters, also known as kernels, which are essentially small matrices of values known as weights that perform a convolution operation on an image region as they slide across the image. The convolution operation is an element-wise dot product of the kernel matrix and the matrix of pixel values representing the image region. The size of the kernel and the pixel distance the kernel travels, known as the stride, can be defined by a user. The result of each convolution operation is a feature map that amplifies certain features of the image, whether it be the edges of objects or particular patterns. Over time, the network learns the correct weights that minimize the error between its predictions and the actual truth data and adjusts them through so-called backpropagation. Activation functions are then applied

after convolution as a means of introducing non-linearity to the network in order to capture the complexity of the data.

Once these feature maps are obtained, they undergo a max pooling operation where the feature maps have their spatial dimensions reduced by pooling nearby pixels. This process has several functions. First of all, it reduces the computational load by reducing the number of network parameters. It also amplifies the strongest features of an image region. Max pooling is also what makes CNNs spatially invariant. Arguably the most important function of this feature map reduction is to discourage overfitting to training data and encourage network generalization.

After this, feature maps are flattened into one-dimensional feature vectors to be fed into the fully connected layer portion of the network. Each neuron in the network is connected to every other neuron. This is where extracted features are combined and associated with a specific label and set of coordinates. The end result will be a label and set of coordinates with an accompanying confidence score, which is the probability that an object within an image actually belongs to its predicted class and set of coordinates. This does not mean that the prediction is correct, however. Wrong predictions can have high confidence scores if the network learns poorly from the data.

5.5 Region-based Convolutional Neural Networks (R-CNN)

R-CNNs are an extension of CNNs that are designed for object localization; that is, the identification of objects in images and where in the image they exist. The first R-CNN was not a single end-to-end trainable network, but rather a pipeline model of multiple stages, as shown in Figure 32. Images processed by an R-CNN are first fed into a region proposal algorithm, typically Selective Search, which identifies RoIs within each image where objects are likely to be. The algorithm selects regions by grouping similar collections of pixels together based on a number of visual criteria.

These extracted regions of interest are passed to a CNN individually. Object features are then identified and extracted as feature vectors. From there, a support vector machine (SVM) would classify the object and refine bounding box proposals through regression analysis [11].

The issue with this method is that the need to process RoIs individually creates a significant bottleneck, as there could be hundreds or even thousands of region proposals for each image. However,

future iterations of R-CNNs addressed this particular issue.

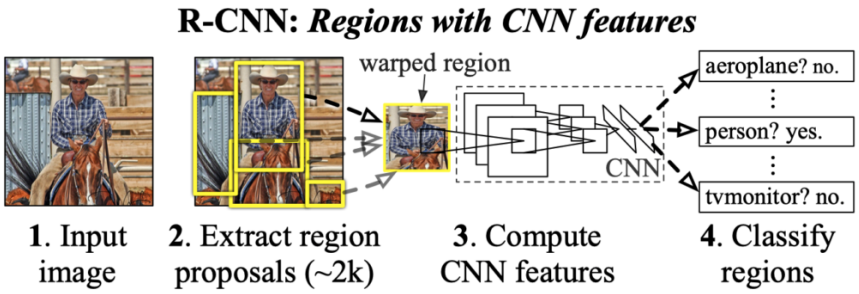


Figure 35: R-CNN image processing pipeline [11]

5.6 Fast Region-based Convolutional Neural Networks (Fast R-CNN)

With the introduction of Fast R-CNN, the region proposal processing bottleneck was resolved. A new addition to the CNN's architecture known as a RoI pooling layer allowed for a significant breakthrough. Instead of processing each RoI individually, the CNN would process the entire image once and the extracted RoIs would be fed into the RoI pooling layer alongside the image feature map to get fixed-length feature vectors for each region.

RoI pooling is a type of max pooling operation that takes the input region, matches it with its corresponding section of the image feature map, and converts that section into a fixed-length feature vector through pooling. These feature vectors are passed to fully connected layers for classification and bounding box regression. Since the same feature map can be used for each region proposal, the image only needs to be processed once by the CNN, and computation for overlapping regions can be shared [12]. This gives Fast R-CNN a considerable speed advantage of around one order of magnitude compared to R-CNN.

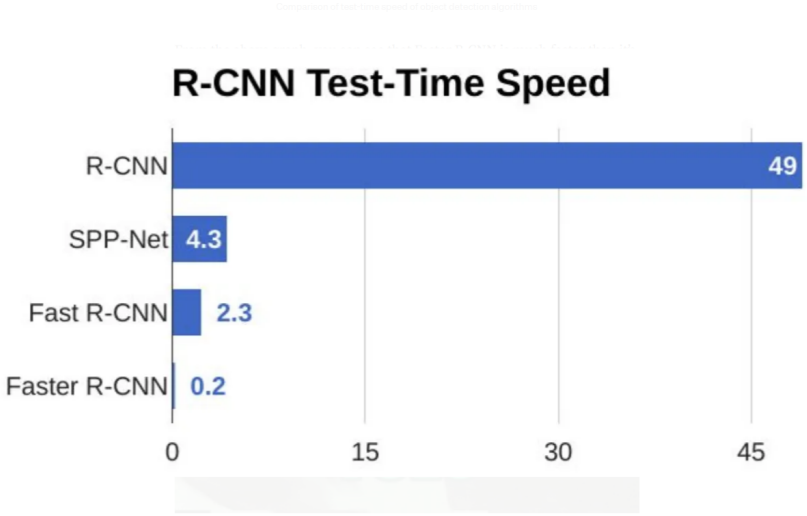


Figure 36: R-CNN vs. Fast R-CNN vs. Faster R-CNN testing speed comparison [55].

5.7 Faster Region-based Convolutional Neural Networks (Faster R-CNN)

Unfortunately, Fast R-CNN still retains the drawback of depending on an external algorithm for identifying RoIs. Algorithms like Selective Search are constrained to run on CPUs, creating another computational bottleneck. Faster R-CNN aimed to solve this issue by adding a new structure to the network's architecture and eliminating the need for an external algorithm entirely. This new addition was a small network attached to the main CNN backbone called the RPN. The RPN takes over the role of identifying regions of interest and it accomplishes this using a secondary CNN [13]. With this new addition, the algorithm could also now be trained on GPUs and capitalize on their parallel processing capabilities to increase the network's training speed.

The main CNN backbone of the network processes an image and generates a feature map. The RPN then takes the feature map and generates anchor boxes for each point in the feature map, which are referred to as anchor points. The intersection-over-union (IoU), or overlap, of these anchor boxes with the ground truth label boxes given by the training dataset is calculated. If the overlap is calculated to be over a certain threshold value, the network identifies this as a region of interest. The benefit of having region proposals generated this way is that now the entire network is an end-to-end model that can be trained entirely using GPU parallel processing. This makes image processing an order of magnitude faster than Fast R-CNN. A comparison of runtime speeds for the

R-CNN, Fast R-CNN, and Faster R-CNN architectures is shown in Figure 33. For SSA applications, Faster R-CNN's image processing speed makes it suitable for real-time object detection and can potentially be integrated with a satellite payload's onboard systems for active space-based object tracking.

5.8 Advantages of Faster R-CNN in SSA

There are some advantages of using CNN-based algorithms for object detection over traditional computer vision algorithms such as frame differencing algorithms and optical flow analysis. In SSA applications, the most critical advantage is that CNNs are spatially invariant by design. This comes from the fact that precise object position information is discarded through max pooling. In simple terms, this means that CNNs are less sensitive to object translations than other computer vision algorithms. While not unique to Faster R-CNN, this advantage would carry over given that Faster R-CNN is a convolutional network.

Despite being slower than YOLO, Faster R-CNN does have certain advantages over YOLO in SSA applications and still maintains a considerable speed advantage over previous R-CNN versions, being two orders of magnitude faster than R-CNN and around one order of magnitude faster than Fast R-CNN in testing time, respectively. When it comes to detecting small objects, such as RSOs, YOLO has more difficulty due to its deep architecture. In particular, the problem arises from the max pooling layers. Generally, as images are processed through convolutional layers, their sizes are reduced because feature map outputs are max pooled. This can cause small object features to disappear from the feature map outputs during forward propagation somewhere in the hidden layers [42]. This means that YOLO risks losing visual information since the most basic YOLO configurations include multiple max pooling layers. Faster R-CNN, on the other hand, does not employ traditional max pooling and its CNN backbone can be customized to meet the specific requirements of the dataset.

6 Faster R-CNN for RSO Detection and Classification

6.1 Datasets

Fast Auroral Imager (FAI) is the optical payload aboard the Canadian Space Agency's CASSIOPE small-satellite spacecraft, which was designed to capture aurora [43]. FAI images were chosen for this study because they contain various RSOs of interest, and FAI itself has a wide FOV comparable to those of star tracker cameras. The wide FOVs of star tracker cameras make them an ideal choice for object detection in a sky background, and, if repurposed for such a task, provide a convenient avenue through which to greatly expand RSO tracking capabilities due to their widespread use across a large number of spacecraft [44]. FAI is a low-resolution sensor but it has large pixels, which is an asset when trying to image auroral emissions in low-light conditions. Sensors are able to produce images based on the interaction of photons with the surface of the sensor, which is an array of pixels. Pixels accumulate charge based on the number of photons striking the detector, as a consequence of the photoelectric effect. The photoelectric effect describes the emission of electrons from a material such as silicon, caused by collisions from electromagnetic radiation. Incoming photons absorbed by the material can cause electrons to be ejected, depending on the energy imparted by the photon interaction. Dislodged electrons accumulate across the pixel array as electric charges which are then converted into a digital signal to be interpreted as an image. The number of pixels passing through an area in a given time is known as the photon flux.

$$\Phi = \frac{N}{At} \tag{24}$$

The ability for a sensor to detect objects depends on the physical limitations of the sensor, as well as the photon flux. A sensor may have a very high resolution for instance, but due to a smaller pixel size it may not be able to accumulate enough charge from photon interactions to capture an image distinct from background noise. This is where FAI's low-resolution, large-pixel construction becomes particularly useful. The larger pixels mean that even in low-light conditions, where the number of incoming photons is limited, the pixels can still capture enough of them to produce an

image.

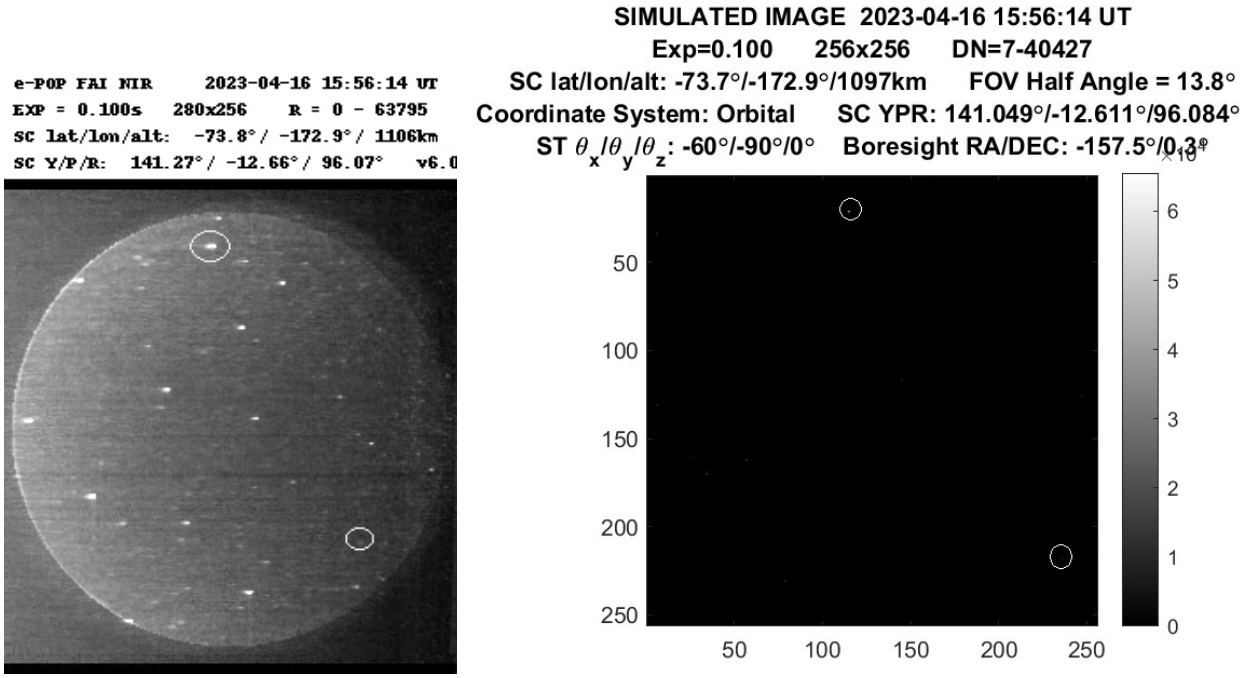
Due to issues with CASSIOPE's reaction wheels, FAI is currently stuck in an anti-sun pointing direction. This is an advantageous sensor orientation for viewing RSOs in various orbits, as there is far less interference from solar glare.

Another advantage of using FAI is that all images are publicly available. This allows anyone to download an extensive dataset of star field images under different orbital and lighting conditions as needed by the study of interest. Machine learning requires large datasets to successfully train an algorithm. With this dataset, there is a large and readily available surplus of images. The images just have to be labeled to be useful for training.

This Faster R-CNN implementation was trained on two different datasets, described in the subsequent subsections. The first dataset the algorithm was trained on consisted of simulated FAI images, while the second consisted of real processed FAI images. The simulated dataset was used to test Faster R-CNN's small object detection capabilities.

6.2 Simulated Data

The simulated FAI image dataset was generated using SBOIS, first introduced in [30] and developed in the York University Nanosatellite Laboratory. SBOIS allows users to generate an extensive dataset of images that can be used for object detection testing. The reason for this is that in the process of generating simulated images, the positions of RSOs visible from the target are recorded to generate the images. Additionally, the positions of the stars and RSOs in the images are recorded in a separate file. This allows users to obtain truth labels for training without having to manually label the images, unlike with real satellite images. In [30], the authors prove that the simulated images demonstrate a realistic representation of stars and RSOs when compared to a functionally similar commercial tool, such as STK, or the real images captured from FAI. A side-by-side comparison between a real and simulated FAI image generated by SBOIS is shown in figure 37.



(a) Real FAI Image

(b) Simulated FAI Image

Figure 37: Comparison between real and simulated FAI image on 15:45:14 April 16 2024 UTC. Two white circles are placed around a couple of RSOs for location comparison. It should be noted that the RSOs are all modeled as 10 cm Lambertian spheres which is why they do not appear as bright as the RSOs in the real image. Due to the image being scaled in brightness, the stars appear very faint.

The simulated images used for this study were scaled up from a series of 640x640 resolution images to 1280x1280 images. These images were previously used in a YOLO study and were scaled up by the original author of that study to attempt to improve performance when using the algorithm. The simulated training image set consisted of 449 grayscaled images. The validation set consisted of 60 images. Prior to use, the images underwent brightness thresholding to improve the visibility of dimmer objects. It is important to note that there is only one object class. The original author of the annotated dataset did not make a distinction between stars and RSO classes in the prior YOLO study. Still, this dataset was considered useful as a study of the effectiveness of Faster R-CNN for small object detection.

6.3 Real Data

The second dataset used for this study was comprised of real FAI images that had undergone pre-processing. These images were acquired from the University of Calgary's website for their CASSIOPE e-POP payload. The pre-processing component consisted of several steps. These were performed on rolling sets of three grayscale images. The first step was to apply a uniform threshold to all three images to remove background noise. Persistent lit pixels, which were a product of radiation effects, were subsequently removed by converting the images to lit pixel lists. These lit pixel lists would then be compared to one another, and any pixels that were consistently bright across all three images would be removed. To reduce the number of false positive RSO detections, the majority of stars (but not all) were then removed from the images. This was done by predicting where any fixed objects (i.e. stars) would be in each image, then using the previous image and the spacecraft's known attitude change between images to subtract any lit pixels present in the predicted areas. After the star removal process was complete, the three grayscale images were combined into a single RGB image, with each image being represented by a different color channel. Finally, a Gaussian convolution was applied to magnify the RSO features present in the processed image to facilitate their detection by the Faster R-CNN [41].

These images are 256x280 resolution, with accompanying ground truth annotations stored as the bottom left corner coordinates and the width and height of each box. Objects are classified as either stars or RSOs in this dataset. The dataset has a random 80/20 training/validation split. In this dataset, there are 1244 training images and 311 validation images. It is important to note that this dataset was manually labeled and, as such, perfect accuracy and consistency in object labeling could not be guaranteed despite painstaking efforts.

In addition to potential labeling errors, the pre-processing technique was not without flaws. Even though the pre-processing significantly reduced the number of false positive RSO detections, there are several areas where pre-processing improvements could be made. When performing image thresholding, a fixed threshold value was used which of course varied in effectiveness depending on the different image lighting conditions and could be improved using adaptive thresholding methods. Adaptive methods could lead to significant improvement under different lighting conditions, rather

than applying a constant threshold throughout. Furthermore, the star removal technique also removed slow-moving RSOs from the images and was highly dependent on the accuracy of the spacecraft's attitude estimates. Replacing the current star removal method with an Iterative Closest Point (ICP) method could reduce the number of RSOs removed during pre-processing while also removing more stars and by extension would improve the Faster R-CNN's performance.

6.4 Methodology

The Python libraries selected for this custom Faster R-CNN implementation were PyTorch and TorchVision, with TorchVision providing the required Faster R-CNN operators. The network was trained on a PC with an Intel i7-11700F CPU, 16GB RAM, and an NVIDIA RTX 3080 GPU with 10GB VRAM. The network was trained on GPU.

The network itself consists of a basic sequential backbone CNN made up of two zero-padded convolutional layers using default 3x3 kernels, and rectified linear unit (ReLU) activation layers. ReLU activation is the standard in most CNN applications as it leads to almost universally better performance than other types of activation functions [45]. This small backbone configuration was chosen to have enough complexity to learn the features of the low-resolution RSO images without being too large to potentially run on satellite onboard hardware if future flights were a consideration or a possibility.

Layer	**In Channels**	**Out Channels**	**Kernel**	**Stride**	**Padding**	**Bias**	**Padding Mode**
Conv2D	3	32	3x3	1	1	True	Zeros
ReLu	N/A						
Conv2D	32	32	3x3	1	1	True	Zeros
ReLu	N/A						

Table 4: Network backbone structure.

The other portions of the network are the anchor box generator and the RoI pooler. However, for RoI pooling, RoIAlign is used instead of a standard RoI pooling layer. RoIAlign is a more precise version of RoI pooling because it operates at the subpixel level and avoids data loss from the quantization of data that standard RoI pooling is known for [46]. Anchor box sizes of 4, 8, and 16 were chosen because these were in line with the sizes of the RSOs and stars in the dataset in terms of pixel dimensions. The aspect ratios are default. The anchor box sizes and aspect ratios chosen for each dataset are depicted in Table 5. The RoI pooler output size was chosen to be 7 based on

output feature map dimensions discussed in [12], while the sampling ratio was chosen based on the sampling ratio shown in [47]. The sampling ratio refers to the dimensions of the grid used when subdividing feature map grid cells and pooling fractional pixels.

Dataset	Sizes	Aspect Ratios
Simulated	(32, 64, 128) and (8, 16, 32)	(0.5, 1, 2)
Real	(4, 8, 16)	(0.5, 1, 2)

Table 5: Anchor box generator settings.

Feature Map Names	Output Size	Sampling Ratio
["0"]	7	2

Table 6: RoI pooler settings.

Several training sessions were performed on each training dataset. The first dataset was trained on a CPU, while the second dataset was trained using a GPU. The VRAM requirements for the first dataset proved to be too large, so CPU-based training was the only option. Due to concerns regarding memory constraints, a small batch size of 4 was used for training for each dataset. This smaller batch size has an overall positive impact on network learning and leads to faster convergence because the model's weights are updated more frequently, but it can come with the downside of less accurate gradient estimation and a longer training time. In fact, the relationship between batch size and training time is exponential. Initial tests were conducted on the simulated FAI dataset using anchor box sizes of 32x32, 64x64, and 128x128, although these were scaled down to improve results after initial testing.

6.5 Results From Simulated FAI Dataset

After training for only 10 epochs with the aforementioned anchor box sizes, the results were poor, as shown in Figure 35. When performing network predictions on the validation set, a limited number of objects were detected. The loss value remained steady during training, indicating a lack of improvement over time.

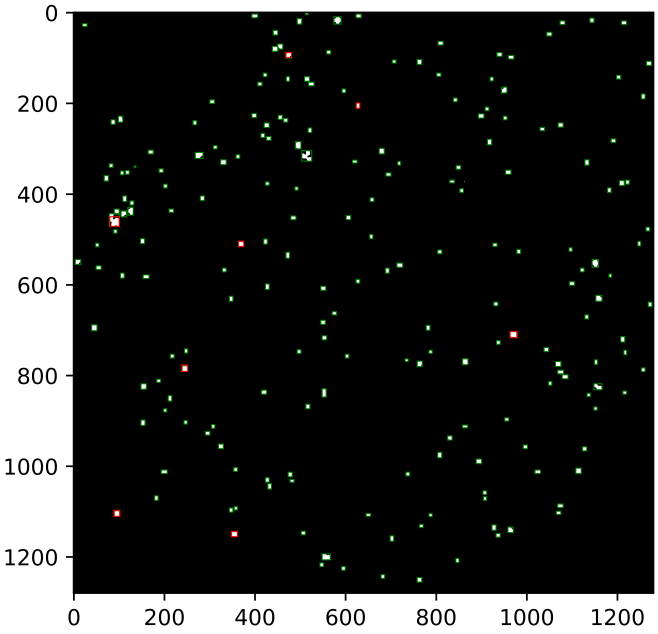


Figure 38: Bounding box predictions (red) after initial training for 10 epochs, with ground truth labels present (green).

It was hypothesized that increasing the radius of the ground truth label boxes may have a positive effect on the training process. Given that the ground truth label boxes were prepared for a YOLO implementation, their conversion to a PyTorch-suitable format appeared to leave the edges of the objects outside of the area of the ground truth annotation boxes. This idea was put to the test with a 100-epoch training cycle on the same dataset with an additional two pixels of spacing added at the edges of the annotation boxes, as shown in Figure 36.

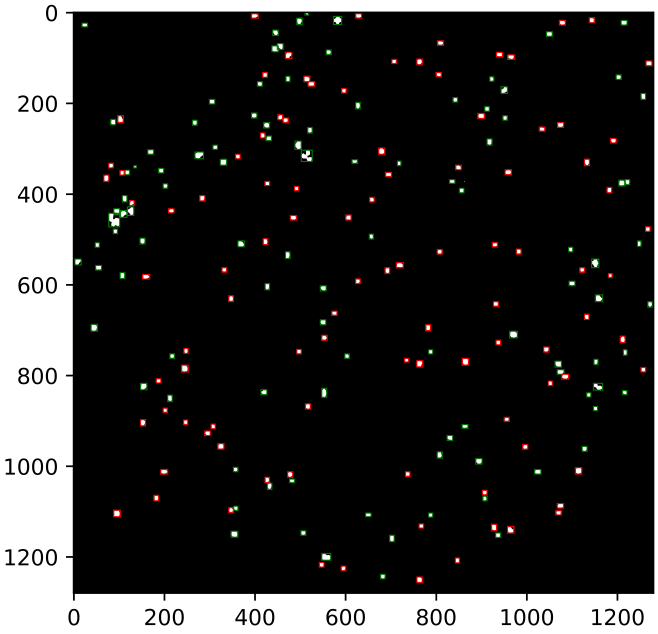


Figure 39: Bounding box predictions (red) after training for 100 epochs with the enlarged label boxes. The number of objects shown to be detected compared to the total present in this particular image is 100/203.

After this, the dataset was trained on the unmodified dataset but with smaller anchor boxes. The bounding box predictions for this scenario are shown in Figure 37. This time the anchor boxes were scaled down from 32x32, 64x64, and 128x128 to 8x8, 16x16, and 32x32 in an effort to better align with the actual sizes of the objects. This had a positive effect on the network's performance, with network prediction results being similar to those of the previous 100-epoch training cycle after only 10 epochs. On average, 48% of objects are detected in the validation set. Precision and recall values were not calculated for this dataset since there is only one object class.

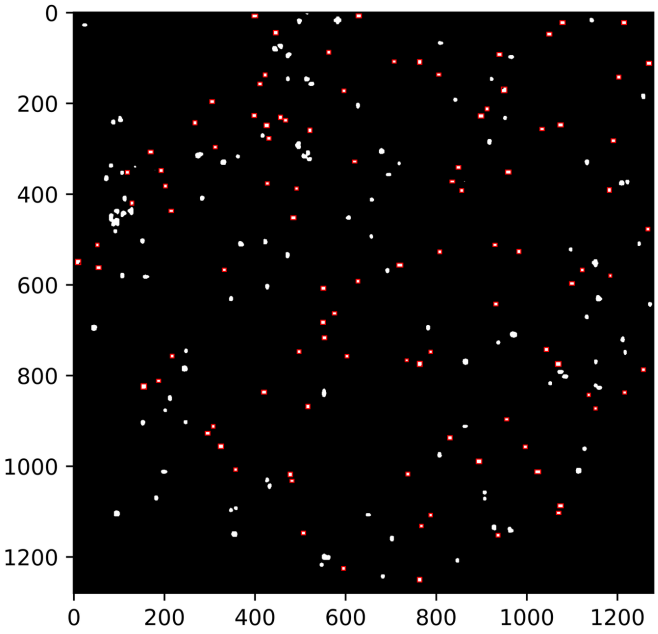


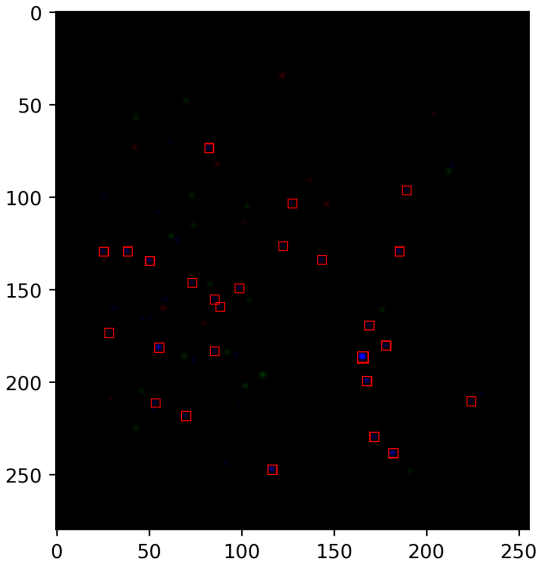
Figure 40: Bounding box predictions (red) after training for 10 epochs with reduced anchor box scales.

The network was subsequently trained for a further 25 and then 30 epochs but results did not improve. Object detection remained at around 48%. Loss decreased steadily, but accuracy remained the same. This suggests overfitting. This is when the model corresponds too closely with the training dataset and is unable to properly generalize to unseen data. This prompted the exploration of altered network structures, different batch sizes, and changes to other hyperparameters.

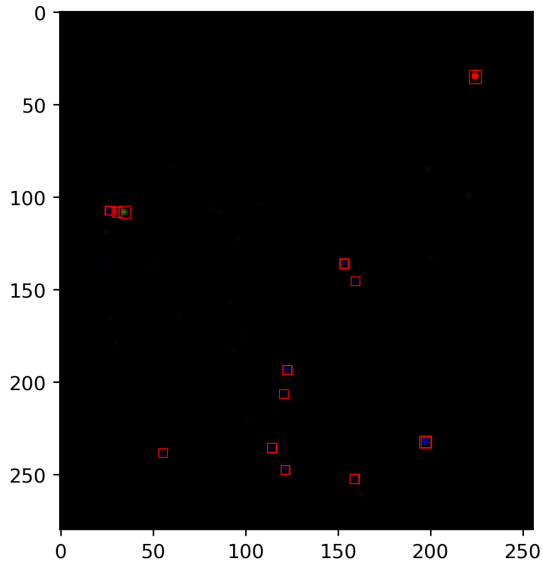
A few different configurations and hyperparameter changes were explored in order to improve detection results. First, a network configuration with only one convolutional layer was attempted, but this yielded the same results. A configuration with 3 convolutional layers led to a reduction in average detections from around 48% down to around 46%. Increasing the batch size to 8 resulted in a significant reduction in training time, but saw no increase in accuracy. Increasing the weight decay parameter from 0.0005 to 0.001 also did not change anything. This was also tried in combination with "early stopping," which is a technique that can help prevent overfitting, but this did not lead to improvements either. There are several other techniques that can be utilized to help prevent overfitting, but these were not applied to the work with this dataset as it was not the main focus of this study.

6.6 Results With Real FAI Dataset

The next step was to apply this network structure to the real FAI dataset that had undergone significant preprocessing. Since the objects in these images have a smaller radius than those of the simulated dataset, it was considered to be suitable to reduce the anchor box scales to 4x4, 8x8, and 16x16. Prediction results were obtained on the validation set after 100 epochs, 500 epochs, and 1000 epochs, as shown in Figures 38-40, respectively.

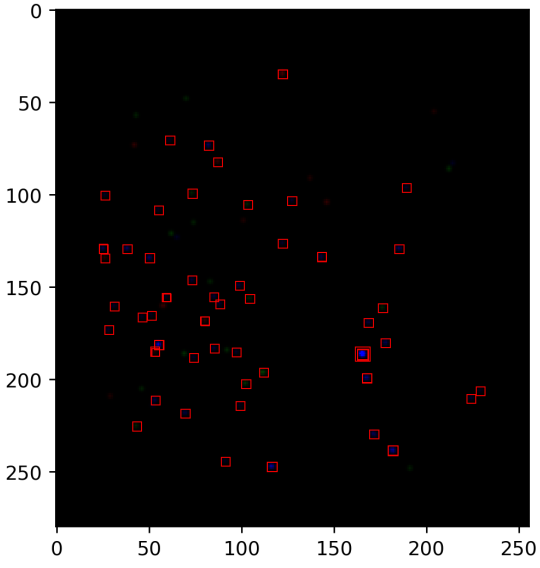


(a) Image 1

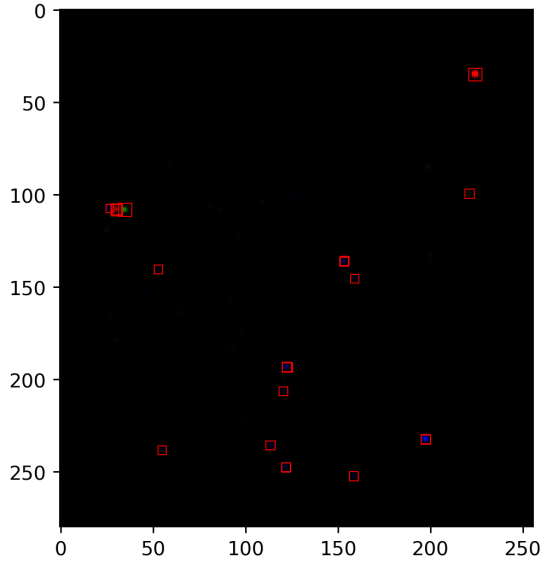


(b) Image 2

Figure 41: Network prediction results on real FAI validation images after 100 training epochs.



(a) Image 1

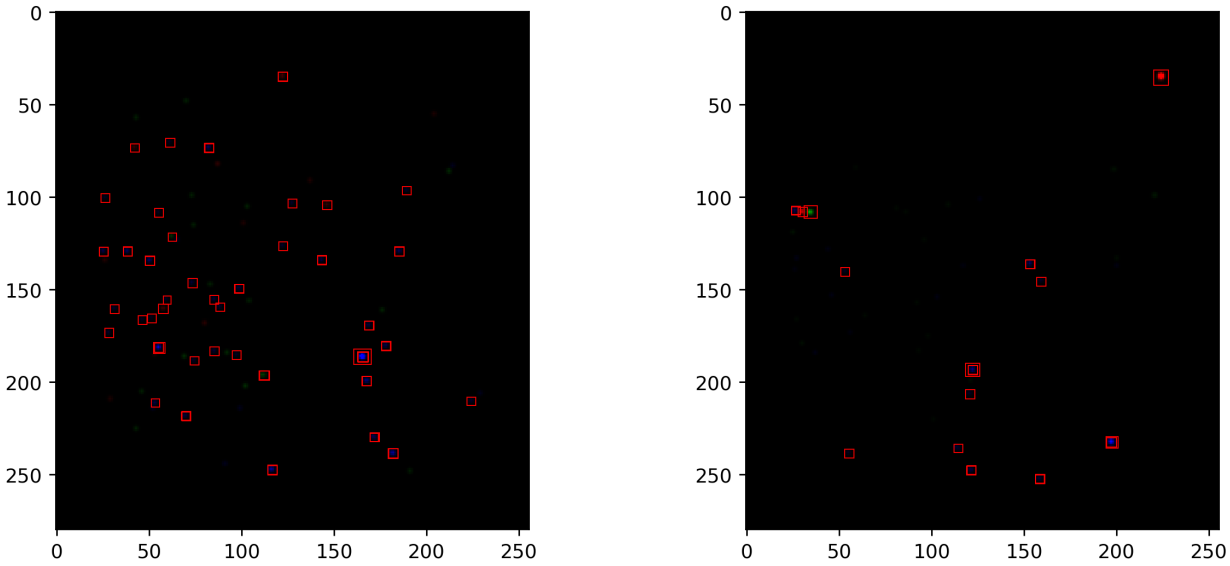


(b) Image 2

Figure 42: Network prediction results on real FAI validation images after 500 training epochs.

There is an obvious improvement in the number of brighter objects detected between 100 and

500 training epochs. Very faint objects are consistently missed. The images on the right for both the 100 and 500 epoch examples demonstrate this. At 1000 epochs, detections are actually lost, which would suggest overfitting.



(a) Image 1

(b) Image 2

Figure 43: Network prediction results on real FAI validation images after 1000 training epochs.

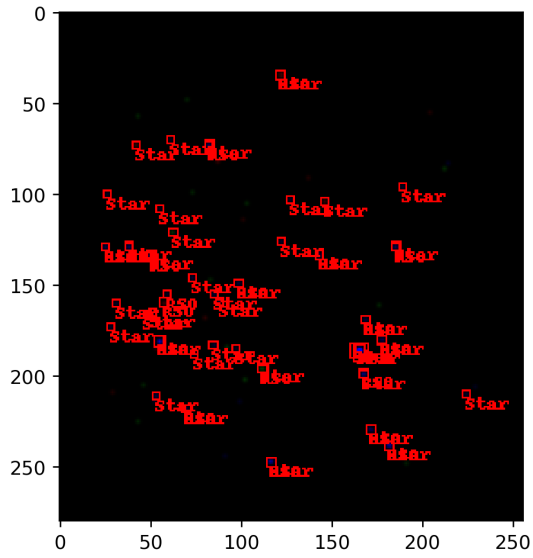


Figure 44: Network predictions after 1000 epochs for figure 40(a) with object labels.

Object classification performance was not clear at first, since many objects had multiple prediction boxes. This could potentially be fixed by adjusting the non-max suppression (NMS) threshold of the neural network before training, as well as the confidence (objectness) or IoU thresholds. For this study, these network parameters were left at their default values. Since many objects were detected more than once and had conflicting classifications in some cases, it made obtaining quantitative performance results difficult, but not impossible. We were able to evaluate network performance by considering classifications corresponding to bounding boxes that had the highest confidence scores. Figure 41 shows the classification network predictions after 1000 epochs.

Classification performance of the network after 500 training epochs (the milestone where performance appeared to be best) was obtained for each of the 311 validation images. The micro-averaged precision and recall were calculated in batches. When calculating micro-averaged precision and recall, both object classes are included in the calculation, rather than calculating these metrics separately for each class.

In object detection contexts, true positives are objects that are detected and assigned correct labels. False positives are objects that are detected but assigned an incorrect classification label. Finally, we have false negatives, which are instances where the network fails to detect a known object

entirely. In the context of object detection, precision is the ratio of correctly classified objects to the total number of detected objects. Recall is the ratio of correctly classified objects to the number of relevant objects to detect. The micro-averaging approach to calculating these metrics helps to take into consideration the effects of an imbalanced dataset, where we have a predominance of one class over another. Table 7 shows the micro-averaged precision, recall, and F1 score after 500 epochs.

Batch	**Precision**	**Recall**	**F1-Score**
Batch 1 (Images 1-50)	88%	97%	92%
Batch 2 (Images 50-100)	89%	80%	84%
Batch 3 (Images 100-150)	89%	92%	90%
Batch 4 (Images 150-200)	87%	87%	87%
Batch 5 (Images 200-250)	87%	76%	81%
Batch 6 (Images 250-311)	87%	92%	89%
All (Image 1-311)	88%	87%	87%

Table 7: Micro-averaged precision, recall, and F1-score for each batch of images after 500 training epochs.

The overall precision and recall were also calculated at an IoU threshold of 50%. The precision at an IoU of 50% was 85% and the recall was 82%. This means that if we only consider predictions where the overlap of ground truth with prediction boxes was 50% or higher, then 85% of predictions were accurate, while 82% of detected objects had a 50% IoU or higher.

Evidently, the algorithm performed consistently in terms of precision scores. Recall scores are less consistent between image batches, which could be attributed to some batches containing images with certain visual conditions that made object detection more difficult. These would have contributed significantly to the total false negatives and brought the score down overall.

There is no widely agreed-upon threshold for good precision and recall scores. It largely depends on the context. Generally, scores of higher than 80% are considered "good." The exact threshold for what constitutes good scores in a space debris detection context is fairly nebulous without establishing the exact way in which a detector would be used. With these scores, less than 20% of objects would be missed or incorrectly classified by the detector. Whether or not this is acceptable depends on the stakes involved. In an SSA context, recall is likely the more important metric, since it is a measure of how many objects are caught by the detector in the first place. Compared to the 32% recall for Faster R-CNN achieved in [41], this study demonstrates a significant improvement. If

80% is considered a high enough score for deployment on onboard satellite hardware, then this study represents a success in finding a suitably lightweight detection and classification model capable of real-time RSO detection with demonstrably strong performance. In the future, better results could potentially be achieved by tweaking parameters such as the IoU threshold, NMS threshold, anchor box sizes, and anchor box aspect ratios.

Overall, the network performed much better than expected on this data. This is particularly unexpected, given that these are low-resolution star field images and as such any RSOs are depicted as small point source objects. The results demonstrate high network precision and recall above 80%. This study demonstrates that Faster R-CNN has the potential to be an effective, lightweight tool for both object detection and classification in real-world SSA applications and warrants further study.

7 Final Remarks

7.1 Summary

In this section, Faster R-CNN was applied to both simulated and real satellite images to study the suitability of the algorithm for RSO detection, with the need for an autonomous, lightweight, real-time object detection model in mind for potential mission hardware implementation. It is already known that this object detection algorithm can process images in real-time with the right hardware, that being a high-end GPU with an abundance of VRAM. The main purpose of this study was to determine whether or not it could learn enough distinguishing features from low-resolution star tracker-like images so as to effectively localize and classify RSOs. In this regard, the study demonstrated promising results. Star tracker cameras are present onboard many satellites for attitude determination and control, meaning that we can greatly improve SSA by making use of star trackers on satellites that are otherwise not specifically designed for SSA. Star trackers are typically widely available, commercial off-the-shelf sensors and are relatively cheap, meaning that they can be deployed extensively in smaller SSA missions, such as CubeSat missions. Faster R-CNN would be suitable for pairing with star trackers in such missions, as the typical frame rate of a star tracker is anywhere from 1 Hz to 10 Hz. Faster R-CNN can process an image every 100-200 milliseconds, which would be real-time speeds when paired with star tracker hardware.

Although the algorithm performed poorly at detection on the simulated data for reasons that were not explored further within the scope of this research, it performed well on the real, preprocessed image data in terms of both localization and classification. Although there is no widely agreed upon threshold for what constitutes "good" performance in an object detection model, a score of 80% or higher in precision or recall can generally be considered acceptable for reliable implementation in real-world contexts. It is a common threshold in autonomous driving applications. It can be concluded that, based on the performance of the model and its speed advantages on suitable hardware, it is a promising choice for real-time detection and hardware implementation in SSA missions and warrants further study.

7.2 Future Work

A future study with a much larger dataset and a deeper examination of network structure, parameters, and hyperparameters would be the next logical step in conclusively determining the efficacy of R-CNNs for real-world RSO detection. It may be worth exploring the effects of additional layers and increasing the number of output channels with each layer. Increased hyperparameter tuning may also yield even better results. In this study, hardware was a significant limitation and restricted the image batch size used for training. Whether smaller batches ultimately had a positive or negative impact on training is difficult to determine, because the hardware limitations made it impossible to experiment with larger batch sizes. It may be worth training the network on raw images as well.

In a future study, raw images could be used to explore the robustness of this computer vision framework under challenging visual conditions, such as added noise. It would also be interesting to test the network's performance on raw images and consider additional classes beyond stars and RSOs, such as stray light and other visual artifacts as mentioned in [38]. More importantly, however, would be a comparative study with notable AI object detection algorithms such as a lightweight version of YOLO or the efficient EfficientDet algorithm to put Faster R-CNN's performance into perspective among the broader landscape of object detection algorithms.

One significant drawback of the study performed in this thesis was the dataset. The dataset annotations were created under heavy time constraints as part of a feasibility study conducted by Magellan Aerospace. This created difficulties in evaluating the performance of the trained model, as significant data manipulation had to be performed to extract meaningful metrics. Certain metrics such as average precision and mean average precision had to be omitted from the final results, leaving a less clear idea of model performance. In a future study, the dataset would be much more carefully curated and annotated.

8 Contribution

The research presented in this thesis explores the application of machine learning to two challenging areas of SSA. These are the areas of autonomous sensor tasking and object detection. These are areas of SSA that are profoundly impacted by the ever-increasing surplus of RSOs found in our orbit and also greatly benefit from increased automation through the use of intelligent algorithmic approaches.

In the case of sensor tasking, DRL offers an approach to automated scheduling of RSO observations that self-corrects and optimizes its solutions through trial and error via deep learning methods. While this does not produce optimal solutions in all cases, this powerful tool easily rivals the heuristic-based optimization algorithms and manual scheduling methods that have been used in the past. The ability to learn from environmental reward feedback and generalize approaches to scheduling through the adjustment of internal network parameters, coupled with the ability to handle higher dimensional image data, as well as the relative simplicity of implementation through the use of simulation environments and simple reward schemes, makes DRL a topic worth exploring further in the context of SSA. There are undoubtedly excellent use cases for the algorithm in various SSA sensor tasking scenarios, as supported by the scenarios studied in this thesis.

When it comes to object detection and classification, there is a strong need for CNN-based algorithms that can process the kind of low-resolution star tracker-like images produced by many commercial satellites and identify and classify RSOs within them accurately. Dedicated space debris surveillance satellites are expensive and cannot be produced in the significant volume required to keep pace with the growing number of RSOs launched into orbit. Pairing robust automated object detection algorithms like Faster R-CNN with commercial hardware could greatly expand our space debris surveillance capabilities while reducing cost and increasing the speed at which we can meet our RSO tracking needs.

Overall, while not an exhaustive or entirely conclusive study of these algorithms, this research presents the case for the continued exploration of these AI techniques as solutions to difficult SSA challenges.

References

[1] Garcia, Mark. "Space Debris and Human Spacecraft." Brewminate, 30 Jan. 2020, brewminate.com/space-debris-and-human-spacecraft/.

[2] Scharping, Nathaniel. "The Future of Satellites Lies in the Constellations." Astronomy Magazine, 30 June 2021, www.astronomy.com/space-exploration/the-future-of-satellites-lies-in-the-constellations/.

[3] Nicholas, Johnson. "The Collision of Iridium 33 and Cosmos 2251: The Shape of Things to Come." 60th International Astronautical Congress. No. JSC-CN-18971. 2009.

[4] "Astromaterials Research & Exploration Science." NASA Orbital Debris Program Office, NASA, orbitaldebris.jsc.nasa.gov/faq/. Accessed 6 July 2024.

[5] "Space Environment Statistics." Space Environment Statistics · Space Debris User Portal, ESA, sdup.esoc.esa.int/discosweb/statistics/. Accessed 6 July 2024.

[6] R. S. Jakhu, "Iridium-cosmos Collision and its Implications for Space Operations," Yearbook on Space Policy, pp. 254–275, 2010. doi:10.1007/978-3-7091-0318-0_10

[7] "Treaty on Principles Governing the Activities of States in the Exploration and Use of Outer Space, Including the Moon and Other Celestial Bodies." International Legal Materials 6.2 (1967): 386–390. Web.

[8] Ackermann, Mark R., et al. "A Systematic Examination of Ground-based and Space-based Approaches to Optical Detection and Tracking of Artificial Satellites." No. SAND2015-3726C. Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), 2015.

[9] Chen, Lei. "Curse of Dimensionality". Encyclopedia of Database Systems, edited by Ling Liu and M. Tamer Özsu, Springer US, 2009, pp. 545–546, https://doi.org10.1007/978-0-387-39940-9_133.

[10] Redmon, Joseph, et al. "You Only Look Once: Unified, Real-Time Object Detection". 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788, https://doi.org10.1109/CVPR.2016.91.

[11] Girshick, Ross, et al. “Region-Based Convolutional Networks for Accurate Object Detection and Segmentation”. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, 2016, pp. 142–158, https://doi.org10.1109/TPAMI.2015.2437384.

[12] Girshick, Ross. “Fast R-CNN”. 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440–1448, https://doi.org10.1109/ICCV.2015.169.

[13] Ren, Shaoqing, et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, 2017, pp. 1137–1149, https://doi.org10.1109/TPAMI.2016.2577031.

[14] Roberts, Thomas G., et al. “A Deep Reinforcement Learning Application to Space-based Sensor Tasking for Space Situational Awareness.” Proceedings of the 2021 Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS), Wailea Beach Resort, Maui, HI, USA. 2021.

[15] Jang, Daniel & Siew, Peng Mun & Gondelach, David & Linares, Richard. (2020). “Space Situational Awareness Tasking For Narrow Field Of View Sensors: A Deep Reinforcement Learning Approach.”

[16] Little, Bryan D., and Carolin Frueh. “SSA Sensor Tasking: Comparison of Machine Learning with Classical Optimization Methods.” Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference. 2018.

[17] B. Oakes, D. Richards, J. Barr, and J. Ralph, “Double Deep Q Networks for Sensor Management in Space Situational Awareness,” 2022 25th International Conference on Information Fusion (FUSION), Linköping, Sweden, 2022, pp. 1-6, doi: 10.23919/FUSION49751.2022.9841242.

[18] “Introduction to Hill Climbing: Artificial Intelligence.” GeeksforGeeks, GeeksforGeeks, 20 Apr. 2023, www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/.

[19] Lin, Wei-Chen, and Da-Yin Liao. “A Tabu Search Algorithm for Satellite Imaging Scheduling.” 2004 IEEE International Conference on Systems, Man, and Cybernetics (IEEE Cat. No. 04CH37583). Vol. 2. IEEE, 2004.

[20] Applegate, David L. “The Traveling Salesman Problem: A Computational Study.” Vol. 17. Princeton University Press, 2006.

[21] Herz, Alex, et al. “Combined SSA Sensor Tasking for Space-to-Space and Ground-to-Space.” Advanced Maui optical and space surveillance technologies conference (AMOS). 2017.

[22] Zhai, Xuejun, et al. “Robust Satellite Scheduling Approach for Dynamic Emergency Tasks.” Mathematical Problems in Engineering, vol. 2015, no. 1, 2015, p. 482923, https://doi.org/10.1155/2015/482923. Accessed 7 Jul. 2024.

[23] Globus, Al, et al. “A Comparison of Techniques for Scheduling Earth Observing Satellites.” AAAI. 2004.

[24] Wolfe, William J., and Stephen E. Sorensen. “Three Scheduling Algorithms Applied to the Earth Observing Systems Domain.” Management Science 46.1 (2000): 148-166.

[25] Sutton, Richard S., and Andrew G. Barto. “Reinforcement learning: An Introduction.” MIT Press, 2018.

[26] Ng, Ritchie. “Markov Decision Processes (MDP) and Bellman Equations.” Markov Decision Processes (MDP) and Bellman Equations - Deep Learning Wizard, www.deeplearningwizard.com/deep_learning/deep_reinforcement_learning_pytorch /bellman_mdp/. Accessed 7 July 2024.

[27] “Reinforcement Learning Applications: From Gaming to Real-World.” Deepchecks, 18 Apr. 2023, deepchecks.com/reinforcement-learning-applications-from-gaming-to-real-world/.

[28] Berner, Christopher, et al. “Dota 2 with Large Scale Deep Reinforcement Learning.” arXiv preprint arXiv:1912.06680 (2019).

[29] Mnih, Volodymyr, et al. “Human-level Control Through Deep Reinforcement Learning.” nature 518.7540 (2015): 529-533.

[30] Clark, Ryan, et al. “Simulation of RSO Images for Space Situation Awareness (SSA) Using Parallel Processing.” Sensors 21.23 (2021): 7868.

[31] Suthakar, Vithurshan, et al. “Comparative Analysis of Resident Space Object (RSO) Detection Methods.” Sensors 23.24 (2023): 9668.

[32] Zuehlke, David, et al. “Autonomous Satellite Detection and Tracking Using Optical Flow.” arXiv preprint arXiv:2204.07025 (2022).

[33] Tan, Mingxing, Ruoming Pang, and Quoc V. Le. “EfficientDet: Scalable and Efficient Object Detection.” Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[34] Tan, Mingxing, and Quoc Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.” International conference on machine learning. PMLR, 2019.

[35] Duan, Kaiwen, et al. “CenterNet: Keypoint Triplets for Object Detection.” Proceedings of the IEEE/CVF international conference on computer vision. 2019.

[36] Tao, Jiang, et al. “Deep Convolutional Neural Network Based Small Space Debris Saliency Detection.” 2019 25th International Conference on Automation and Computing (ICAC). IEEE, 2019.

[37] De Vittori, Andrea, et al. “Real-Time Space Object Tracklet Extraction from Telescope Survey Images with Machine Learning.” Astrodynamics 6.2 (2022): 205-218.

[38] Lim, Michael, et al. “Onboard Artificial Intelligence for Space Situational Awareness with Low-Power GPUs.” Advanced Maui Optical and Space Surveillance Technologies Conference. 2020.

[39] AlDahoul, Nouar, et al. “Localization and Classification of Space Objects Using EfficientDet Detector for Space Situational Awareness.” Scientific Reports 12.1 (2022): 21896.

[40] Yuan, Yuman, et al. “An Intelligent Detection Method for Small and Weak Objects in Space.” Remote Sensing 15.12 (2023): 3169.

[41] Qashoa, Randa, et al. “SPACEDUST-Optical: Wide-FOV Space Situational Awareness from Orbit.” Proceedings of the Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference. 2023.

[42] A. Dolgarev, “Small Objects Detection Problem,” Quantumobile, https://quantumobile.com/blog/small-objects- detection-problem/ (accessed Jan. 17, 2024).

[43] “FAI,” e-POP, https://epop.phys.ucalgary.ca/fai/ (accessed Jan. 17, 2024).

[44] D. Siddharth, R. Lee, “Feasibility of a Virtual Constellation using Small Aperture, Wide Field of View Optical Systems for Space Domain Awareness and Applications.” AMOS Conference, 2022.

[45] J. Brownlee, “A Gentle Introduction to the Rectified Linear Unit (ReLU),” MachineLearningMastery.com, https://machinelearningmastery.com/rectified-linear-activation- function-for-deep-learning-neural-networks/ (accessed Jan. 17, 2024).

[46] K. Patnaik, “Annotated RPN, ROI Pooling and ROI Align,” Kaushik’s Blog, https://kaushikpatnaik.github.io/annotated/papers/2020/07/04/ROI- Pool-and-Align-Pytorch-Implementation.html (accessed Jan. 17, 2024).

[47] N. Sardana, ”Instance Segmentation,” TJHSST Machine Learning Club, Jan. 10, 2018. [Online]. Available: tjmachinelearning.com/lectures/1718/instance/. [Accessed: Dec. 24, 2024].

[48] A. Nair, M. Hessel, O. Vinyals, V. Mnih, and A. Graves, ”Massively Parallel Methods for Deep Reinforcement Learning,” arXiv preprint, arXiv:1507.04296, 2015.

[49] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, ”Proximal Policy Optimization Algorithms,” arXiv preprint, arXiv:1707.06347, 2017.

[50] ”Kepler’s Laws of Planetary Motion,” Physics Bootcamp.[Online]. Available: http://www.physicsbootcamp.org/Keplers-Laws-of- Planetary-Motion.html. [Accessed: Dec. 24, 2024].

[51] ”Free PNG Image,” PNGWing. [Online]. Available: https://www.pngwing.com/en/free-png-kletp. [Accessed: Dec. 24, 2024].

[52] European Space Agency, ”Types of orbits,” ESA - Enabling Support, 2023. [Online]. Available: https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits. [Accessed: 24-Dec-2024].

[53] R. Stottler, ”Intelligent Space Surveillance Network (SSN) Scheduling Applications,” Infotech@ Aerospace 2012, 2012, p. 2434.

[54] Konnik, M., & Welsh, J., "High-level numerical simulations of noise in CCD and CMOS photosensors: review and tutorial," arXiv preprint arXiv:1412.4031, 2014. [Online]. Available: https://arxiv.org/abs/1412.4031.

[55] A. Ali, K. Saikia, B. Nayak, M. K. Muchahari, and P. Kumar, "Augmented reality based online application for e-shopping," Int. J. Adv. Res. Eng. Technol. (IJARET), vol. 12, no. 3, 2021.

[56] H. Uy, "Section 4.3 - The Six Orbital Elements," Astronomical Returns. [Online]. Available: https://www.astronomicalreturns.com/p/section-43-six-orbital-elements.html. [Accessed: Feb. 16, 2025].

[57] R. Qashoa, P. Harrison, and R. Lee, "Wide Field of View (FOV) imagers for co-orbiting object detection," Proceedings of the Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference, 2024. Available: https://www.amostech.com.